Fast transforms over finite fields of characteristic two

Nicholas Coxon

Fast transforms over finite fields of characteristic two*

Nicholas Coxon

INRIA and Laboratoire d'Informatique de l'École polytechnique, Palaiseau, France.

Abstract

We describe new fast algorithms for evaluation and interpolation on the "novel" polynomial basis over finite fields of characteristic two introduced by Lin, Chung and Han (FOCS 2014). Fast algorithms are also described for converting between their basis and the monomial basis, as well as for converting to and from the Newton basis associated with the evaluation points of the evaluation and interpolation algorithms. Combining algorithms yields a new truncated additive fast Fourier transform (FFT) and inverse truncated additive FFT which improve upon some previous algorithms when the field possesses an appropriate tower of subfields.

Keywords: Fast Fourier transform, Newton basis, finite field, characteristic two

1. Introduction

Let $\mathbb{F}$ be a finite field of characteristic two, and $\beta = (\beta_0, \ldots, \beta_{n-1}) \in \mathbb{F}^n$ have entries that are linearly independent over $\mathbb{F}_2$. Enumerate the $\mathbb{F}_2$-linear subspace of $\mathbb{F}$ generated by the entries of β as $\{\omega_0, \ldots, \omega_{2^n-1}\}$ by setting $\omega_i = \sum_{k=0}^{n-1} [i]_k \beta_k$ for $i \in \{0, \ldots, 2^n - 1\}$, where $[\,\cdot\,]_k : \mathbb{N} \to \{0, 1\}$ for $k \in \mathbb{N}$ such that $i = \sum_{k \in \mathbb{N}} 2^k [i]_k$ for $i \in \mathbb{N}$. For $i \in \{0, \ldots, 2^n - 1\}$, define polynomials

$$X_i = \prod_{k=0}^{n-1} \prod_{j=0}^{2^k [i]_k - 1} \frac{x - \omega_j}{\omega_{2^k} - \omega_j} \quad \text{and} \quad N_i = \prod_{j=0}^{i-1} \frac{x - \omega_j}{\omega_i - \omega_j}.$$

Then the definition of the functions $[\,\cdot\,]_k$ implies that X_i has degree equal to i, while it is clear that N_i also has degree equal to i. Letting $\mathbb{F}[x]_\ell$ denote the space of polynomials over $\mathbb{F}$ with degree strictly less than ℓ, it follows that $\{X_0, \ldots, X_{\ell-1}\}$ and $\{N_0, \ldots, N_{\ell-1}\}$ are bases of $\mathbb{F}[x]_\ell$ over $\mathbb{F}$ for $\ell \in \{1, \ldots, 2^n\}$. The former basis was introduced by Lin, Chung and Han (2014), and is referred to hereafter as the Lin–Chung–Han basis, or simply the LCH basis, of $\mathbb{F}[x]_\ell$ associated with β. The polynomials $N_0, \ldots, N_{2^n-1}$ are scalar multiples of the Newton basis polynomials associated with the points $\omega_0, \ldots, \omega_{2^n-1}$, with the scalars chosen such that $N_i(\omega_i) = 1$. However, we simply refer to $\{N_0, \ldots, N_{\ell-1}\}$ as the Newton basis of $\mathbb{F}[x]_\ell$ associated with β. The space $\mathbb{F}[x]_\ell$ also comes equipped with the monomial basis $\{1, x, \ldots, x^{\ell-1}\}$.

In this paper, we describe new fast algorithms for evaluation and interpolation on the LCH basis, and for conversion between the LCH and either of the Newton or monomial bases. These

*This work was supported by Nokia in the framework of the common laboratory between Nokia Bell Labs and INRIA.
Email address: coxon.nv@gmail.com (Nicholas Coxon)

algorithms may in turn be combined to obtain fast algorithms for evaluation and interpolation on the Newton or monomial bases, and for converting between the Newton and monomial bases. The resulting algorithm for evaluation on the monomial basis provides a new additive fast Fourier transform (FFT). The designation as "additive" reflects the fact that FFTs have traditionally evaluated polynomials at each element of a cyclic multiplicative group, whereas the evaluation points $\omega_0, \ldots, \omega_{2^n-1}$ of our FFT form an additive group. To avoid confusion, we refer to algorithms that evaluate over a multiplicative group as multiplicative FFTs hereafter. Additive FFTs have been investigated as an alternative to multiplicative FFTs for use in multiplication algorithms for binary polynomials (von zur Gathen and Gerhard, 1996; Brent et al., 2008; Mateer, 2008; Chen et al., 2017a, 2018; Li et al., 2018), and have also found applications in coding theory and cryptography (Bernstein et al., 2013; Bernstein and Chou, 2014; Chen et al., 2018; Chou, 2017; Ben-Sasson et al., 2017).

Additive FFTs first appeared in the 1980s with the of algorithm of Wang and Zhu (1988), which was subsequently rediscovered by Cantor (1989). For characteristic two finite fields, the Wang–Zhu–Cantor algorithm requires β to be a so-called Cantor basis: its entries must satisfy $\beta_0 = 1$ and $\beta_{k-1} = \beta_k^2 - \beta_k$ for $k \in \{1, \ldots, n-1\}$. The algorithm then takes the coefficients on the monomial basis of a polynomial in $\mathbb{F}[x]_{2^n}$ and evaluates it at each of the points $\omega_0, \ldots, \omega_{2^n-1}$ with $O(2^n n^{\log_2 3})$ additions in $\mathbb{F}$, and $O(2^n n)$ multiplications in $\mathbb{F}$. Gao and Mateer (2010) subsequently improve upon this complexity by describing an algorithm that performs $O(2^n n \log n)$ additions and $O(2^n n)$ multiplications. However, as for the Wang–Zhu–Cantor algorithm, this complexity is only obtained in a limited setting, since a finite field of characteristic two admits a Cantor basis of dimension n if and only if it contains $\mathbb{F}_{2^{2^{\lceil \log_2 n \rceil}}}$ as a subfield (Gao and Mateer, 2010, Appendix A).

The additive FFT of von zur Gathen and Gerhard (1996) removes the restriction that β must be a Cantor basis, allowing the vector to be chosen subject only to the requirement of linear independence of its entries. Their algorithm performs $O(2^n n^2)$ additions in $\mathbb{F}$, and $O(2^n n^2)$ multiplications in $\mathbb{F}$. Gao and Mateer (2010) subsequently describe an algorithm that performs $O(2^n n^2)$ additions and only $O(2^n n)$ multiplications. Bernstein, Chou and Schwabe (2013) in turn generalise the algorithm of Gao and Mateer so that time is not wasted manipulating coefficients that are known to be zero when the polynomial being evaluated by the transform belongs to $\mathbb{F}[x]_\ell$ for some $\ell < 2^n$. They also describe how to replace some multiplications in their algorithm with less time consuming additions. Bernstein and Chou (2014) contribute several more improvements to the algorithm in the case that β is a Cantor basis.

The generalisation of Bernstein, Chou and Schwabe is obtained by reducing to the case $\ell = 2^n$ and disregarding parts of the algorithm that involve manipulating coefficients that are known to be zero. This technique is often referred to as truncation (or pruning (Markel, 1971; Sorensen and Burrus, 1993)). This term is also applied when only part of a transform is computed by performing only those steps of the algorithm that are relevant to the computation of the desired outputs. Truncation is used in FFT-based polynomial multiplication to ensure that running times vary relatively smoothly in the length of the problem. To achieve such behaviour it is also necessary to invert truncated transforms. However, simply examining the output of a truncated transform may not allow its inversion. An elegant solution to this problem is provided by van der Hoeven (2004, 2005) who took the crucial step of augmenting the output with information about known zero coefficients in the input, allowing him to provide a multiplicative truncated Fourier transform together with its corresponding inverse truncated Fourier transform (see also Harvey, 2009; Harvey and Roche, 2010; Larrieu, 2017). While truncated additive FFTs have been investigated (von zur Gathen and Gerhard, 1996; Mateer, 2008; Brent et al., 2008; Bernstein et al., 2013; Bernstein and Chou, 2014; Chen et al., 2017a, 2018; Li et al., 2018), existing methods for their inversion

lack the effectiveness and elegance of the approach introduced by van der Hoeven.

Alongside the introduction of their basis, Lin, Chung and Han (2014) describe fast algorithms for evaluation and interpolation on the basis that yield lower complexities than additive FFTs. Their evaluation algorithm takes the coefficients on the LCH basis of a polynomial in $\mathbb{F}[x]_{2^n}$ together with an element $\lambda \in \mathbb{F}$ and evaluates the polynomial at each of the points $\omega_0 + \lambda, \ldots, \omega_{2^n-1} + \lambda$ with $O(2^n n)$ additions in $\mathbb{F}$, and $O(2^n n)$ multiplications in $\mathbb{F}$. Their interpolation algorithm inverts this transformation with the same complexity. Lin, Chung and Han then demonstrate the usefulness of their "novel" basis by using the algorithms to provide fast encoding and decoding algorithms for Reed–Solomon codes. This application is further explored in the subsequent works of Lin, Al-Naffouri and Han (2016a) and Lin, Al-Naffouri, Han and Chung (2016b), while Ben-Sasson et al. (2018) utilise the algorithms within their zero-knowledge proof system.

Lin, Al-Naffouri, Han and Chung (2016b) additionally consider the problem of converting between the LCH basis and the monomial basis. They provide a pair of algorithms that allow polynomials in $\mathbb{F}[x]_{2^n}$ to be converted between the two bases with $O(2^n n^2)$ additions in $\mathbb{F}$, and $O(2^n n)$ multiplications in $\mathbb{F}$. Moreover, they provide a second pair of algorithms that allow the conversions to be performed with $O(2^n n \log n)$ additions and no multiplications in the case that β is a Cantor basis. Their algorithms use ideas introduced by Gao and Mateer (2010), and they note that combining their algorithms with the evaluation algorithm of Lin, Chung and Han (2014) yields two additive FFTs, one for arbitrary β and one for Cantor bases, that are "algebraically similar" to the two provided by Gao and Mateer. The additive FFT for Cantor bases is applied to the problem of binary polynomial multiplication in a series of papers (Chen et al., 2017a, 2018; Li et al., 2018).

The (standard) Newton basis of $\mathbb{F}[x]_\ell$ associated with ℓ points $\alpha_0, \ldots, \alpha_{\ell-1} \in \mathbb{F}$ consists of the polynomials $\prod_{j=0}^{i-1}(x - \alpha_j)$ for $i \in \{0, \ldots, \ell - 1\}$. If the points have no special structure, then evaluation and interpolation with respect to the basis, and with evaluation points $\alpha_0, \ldots, \alpha_{\ell-1}$, can be performed with $O(\mathsf{M}(\ell) \log \ell)$ operations in $\mathbb{F}$ by the algorithm of Bostan and Schost (2005), where $\mathsf{M}(\ell)$ denotes the cost of multiplying two polynomials in $\mathbb{F}[x]_\ell$. However, both evaluation and interpolation are known to be less expensive by a logarithmic factor for some special sequences of points (Bostan and Schost, 2005; Smarzewski and Kapusta, 2007), e.g., when the points form a geometric progression only $\mathsf{M}(\ell) + O(\ell)$ operations in $\mathbb{F}$ are required. Similarly, algorithms for converting between the Newton basis and the monomial basis have asymptotic complexity belonging to the class $O(\mathsf{M}(\ell) \log \ell)$ in the general case (Gerhard, 2000; Bostan and Schost, 2005), while once again allowing a logarithmic factor to be saved for some special sequences of points (Bostan and Schost, 2005).

The techniques developed for additive FFTs have yet to be applied to evaluation, interpolation and basis conversion problems involving the Newton basis associated with their evaluation points. However, fast algorithms for converting between the monomial and Newton bases, and for evaluation and interpolation with respect to the Newton basis, would find applications in multivariate evaluation and interpolation algorithms (van der Hoeven and Schost, 2013; Coxon, 2019) and systematic encoding algorithms for Reed–Muller and multiplicity codes (Coxon, 2019). The conversion algorithms would also complement algorithms proposed by van der Hoeven and Schost (2013) for converting between the monomial basis and the Newton basis associated with the radix-2 truncated Fourier transform points (van der Hoeven, 2004, 2005), since their algorithms are not suited to finite fields of characteristic two.

Our contribution

The evaluation, interpolation and basis conversion problems considered in this paper all involve the LCH basis. Consequently, we begin in Section 2 by reviewing properties of the basis that are utilised in the development of our algorithms.

In Section 3, we describe algorithms for converting between the Newton and LCH bases. The algorithms follow the divide-and-conquer paradigm that is characteristic of FFTs, and allow conversion in either direction to be performed for polynomials in $\mathbb{F}[x]_\ell \subseteq \mathbb{F}[x]_{2^n}$ with $\ell(\log_2 \ell)/2 + O(\ell)$ additions in $\mathbb{F}$, and $\ell(\log_2 \ell)/2 + O(\ell)$ multiplications in $\mathbb{F}$. Moreover, the algorithms may be implemented so that only $O(\log^2 \ell)$ field elements are required to be stored in auxiliary space, i.e., in the space used by each algorithm in addition to the space required to store its inputs or outputs. We address the difference between our definition of the Newton basis and the standard definition by showing that the algorithms are readily modified to instead convert to and from the standard Newton basis associated with the points $\omega_0, \ldots, \omega_{2^n-1}$, and at the cost of having to perform only $O(\ell)$ additional operations in $\mathbb{F}$.

In Section 4, we use truncation to generalise the algorithms of Lin, Chung and Han (2014) to allow fast evaluation and interpolation with respect to the LCH basis for polynomials in $\mathbb{F}[x]_\ell \subseteq \mathbb{F}[x]_{2^n}$ and with evaluations points of the form $\omega_0 + \lambda, \ldots, \omega_{c-1} + \lambda$ for some $c \in \{1, \ldots, 2^n\}$ and $\lambda \in \mathbb{F}$, where $c = \ell$ in the case of interpolation. Thus, we provide analogues of van der Hoeven's truncated FFT and inverse truncated FFT (van der Hoeven, 2004). The algorithms each perform at most $c\lceil \log_2 \min(\ell, c)\rceil + 2\ell + c + O(\log^2 \max(\ell, c))$ additions in $\mathbb{F}$, and at most $c\lceil \log_2 \min(\ell, c)\rceil/2 + 2\ell + O(\log^2 \max(\ell, c))$ multiplications in $\mathbb{F}$, and may be implement so that only $O(\log^2 \max(\ell, c))$ field elements are required to be stored in auxiliary space.

Combining the algorithms of Sections 3 and 4 allows for fast evaluation and interpolation with respect to the Newton basis. For example, for polynomials in $\mathbb{F}[x]_\ell \subseteq \mathbb{F}[x]_{2^n}$, evaluation and interpolation at the points $\omega_0 + \lambda, \ldots, \omega_{\ell-1} + \lambda$ for some $\lambda \in \mathbb{F}$ can be performed with $3\ell(\log_2 \ell)/2 + O(\ell)$ additions and $\ell \log_2 \ell + O(\ell)$ multiplications. Thus, our algorithms provide new special sequences in $\mathbb{F}$ for which its possible to improve upon the generic complexity of $O(\mathsf{M}(\ell) \log \ell)$ for Newton evaluation and interpolation.

In the final section of the paper, Section 5, we describe algorithms for converting between the monomial and LCH bases. The algorithms assume the β comes equipped with a tower of subfields $\mathbb{F}_2 = \mathbb{F}_{2^{d_0}} \subset \mathbb{F}_{2^{d_1}} \subset \cdots \subset \mathbb{F}_{2^{d_m}} \subseteq \mathbb{F}$ such that $d_{m-1} < n \leq d_m$, and $\beta_i/\beta_{d_t\lfloor i/d_t \rfloor} \in \mathbb{F}_{2^{d_t}}$ for $i \in \{0, \ldots, n-1\}$ and $t \in \{0, \ldots, m-1\}$. Then for polynomials in $\mathbb{F}[x]_\ell \subseteq \mathbb{F}[x]_{2^n}$ such that $2^{d_s} < \ell \leq 2^{d_{s+1}}$ for some $s \in \{0, \ldots, m-1\}$, the algorithms allow conversion in either direction to be performed with at most

$$\frac{\ell \log_2 \ell}{4}\left(\frac{\log_2 \ell}{d_s} + \sum_{t=0}^{s-1}\left(\frac{d_{t+1}}{d_t} - 1\right)\right) + O(\ell \log \ell) \tag{1.1}$$

additions in $\mathbb{F}$, and at most $\ell \log_2 \ell + O(\ell)$ multiplications in $\mathbb{F}$. Moreover, the algorithms may be implemented so as to require only $O(\log \ell)$ field elements to be stored in auxiliary space.

Excluding the trivial case of $\mathbb{F} = \mathbb{F}_2$, where the monomial and LCH bases coincide, it is always possible to take $m = 1$ and $d_m = [\mathbb{F} : \mathbb{F}_2]$. The algorithms then perform at most $\ell(\log_2^2 \ell)/4 + O(\ell \log \ell)$ additions in $\mathbb{F}$, matching the asymptotic complexity obtained by Lin et al. (2016b) for generic β. However, each subfield of degree less than $\log_2 \ell$ in the tower contributes to the performance of the algorithms by reducing the number of additions and multiplications they perform. The cumulative effect of these contributions is demonstrated by considering the family of towers that have quotients d_{t+1}/d_t bounded by a constant $c \geq 2$. For β equipped with

a tower belonging to this family, our algorithms perform $O(\ell(\log \ell)(\log\log \ell))$ additions, matching the bound obtained by Lin et al. for Cantor bases. Consequently, the algorithms provide additional families of β for which it is possible to improve upon the complexity obtained by Lin et al. for the generic case. To take advantage of this improvement, we describe a method of constructing such a vector when provided with its desired dimension n and an appropriate tower of subfields. We also show how to reduce the number of multiplications performed by the algorithms when the tower contains some quadratic extensions by exploiting freedom in the construction.

The bound (1.1) is in $\Omega(\ell(\log \ell)\log\log \ell)$, suggesting that the algorithms of Section 5 are unable to surpass the asymptotic complexity obtained by Lin et al. for Cantor bases. Consequently, when the algorithms are combined with those of Sections 3 and 4, we do not obtain additive FFTs with asymptotic complexities matching those of the fastest multiplicative FFTs, nor do we obtain algorithms for converting between the monomial and Newton bases with asymptotic complexities matching those of algorithms for other special sequences of points (Bostan and Schost, 2005; van der Hoeven and Schost, 2013). Thus, the problem of converting between the monomial and LCH bases is in need of further attention.

2. Properties of the Lin–Chung–Han basis

It follows from the definition of the LCH basis associated with β that

$$X_{2^k} = \prod_{i=0}^{2^k-1} \frac{x-\omega_i}{\omega_{2^k}-\omega_i} \quad \text{for } k \in \{0,\dots,n-1\}.$$

Thus, the roots of X_{2^k} form an $\mathbb{F}_2$-linear subspace of $\mathbb{F}$, generated by $\beta_0,\dots,\beta_{k-1}$. As most work on additive transforms pre-dates the introduction of the LCH basis, it is typical in the literature to study the properties of the subspace (vanishing) polynomials associated with these subspaces. We instead choose to study the properties of the basis polynomials $X_{2^0},\dots,X_{2^{n-1}}$, which are simply scalar multiples of the subspace polynomials: the subspace polynomial of a subspace $W \subseteq \mathbb{F}$ is defined to be $\prod_{\omega\in W}(x-\omega)$. Consequently, the properties of the LCH basis presented in the section are either found in (Lin et al., 2014, 2016a,b), or are analogous to properties of subspace polynomials found in (Cantor, 1989; von zur Gathen and Gerhard, 1996; Mateer, 2008).

An important property of subspace polynomials, which is inherited by $X_{2^0},\dots,X_{2^{n-1}}$, is that they are linearised. A polynomial in $\mathbb{F}[x]$ is $\mathbb{F}_q$-linearised (alternatively, a q-polynomial) if it can be written in the form $\sum_{i=0}^{k} f_i x^{q^i}$ with $f_0,\dots,f_k \in \mathbb{F}$. The following lemma shows that the polynomials $X_{2^0},\dots,X_{2^{n-1}}$ are $\mathbb{F}_2$-linearised, and establishes several additional properties of the LCH basis that are used in the development of our algorithms.

Lemma 2.1. *The following properties hold for $k \in \{0,\dots,n-1\}$:*

1. $X_{2^k+j} = X_{2^k}X_j$ *for* $j \in \{0,\dots,2^k-1\}$,
2. $X_{2^k}(\omega_{2^k i+j}) = i$ *for* $i \in \{0,1\}$ *and* $j \in \{0,\dots,2^k-1\}$,
3. $X_{2^k} = x/\beta_0$ *if* $k = 0$, *and* $X_{2^k} = (X_{2^{k-1}}(x)^2 - X_{2^{k-1}}(x))/(X_{2^{k-1}}(\beta_k)^2 - X_{2^{k-1}}(\beta_k))$ *otherwise,*
4. X_{2^k} *is $\mathbb{F}_2$-linearised,*
5. $X_{2^k}(x+\lambda) = X_{2^k}(x) + X_{2^k}(\lambda)$ *for* $\lambda \in \mathbb{F}$.

Proof. Let $k \in \{0, \ldots, n-1\}$. Then, for $i \in \{0, 1\}$, $j \in \{0, \ldots, 2^k - 1\}$ and $t \in \{0, \ldots, n-1\}$, we have $[2^k i]_t = 0$ and $[j]_t = [2^k i + j]_t$ if $t < k$, and $[2^k i]_t = [2^k i + j]_t$ and $[j]_t = 0$ if $t \geq k$. Thus,

$$X_{2^k+j} = \prod_{t=0}^{n-1} \prod_{s=0}^{2^t[2^k+j]_t-1} \frac{x - \omega_s}{\omega_{2^t} - \omega_s} = \prod_{t=0}^{n-1} \left(\prod_{s=0}^{2^t[2^k]_t-1} \frac{x - \omega_s}{\omega_{2^t} - \omega_s} \right) \left(\prod_{s=0}^{2^t[j]_t-1} \frac{x - \omega_s}{\omega_{2^t} - \omega_s} \right) = X_{2^k} X_j$$

and

$$\omega_{2^k i+j} = \sum_{t=0}^{n-1} [2^k i + j]_t \beta_t = \sum_{t=0}^{n-1} [2^k i]_t \beta_t + \sum_{t=0}^{n-1} [j]_t \beta_t = \omega_{2^k i} + \omega_j$$

for $i \in \{0, 1\}$ and $j \in \{0, \ldots, 2^k - 1\}$. As $\{\omega_0, \ldots, \omega_{2^k-1}\}$ forms an additive group, it follows that

$$X_{2^k}(\omega_{2^k i+j}) = \prod_{s=0}^{2^k-1} \frac{\omega_{2^k i+j} - \omega_s}{\omega_{2^k} - \omega_s} = \prod_{s=0}^{2^k-1} \frac{\omega_{2^k i} + \omega_j - \omega_s}{\omega_{2^k} - \omega_s} = \prod_{s=0}^{2^k-1} \frac{\omega_{2^k i} - \omega_s}{\omega_{2^k} - \omega_s} = i$$

for $i \in \{0, 1\}$ and $j \in \{0, \ldots, 2^k - 1\}$. Therefore, properties (1) and (2) hold.

If $k = 0$, then $X_{2^k} = (x - \omega_0)/(\omega_1 - \omega_0) = x/\beta_0$. If $k \in \{1, \ldots, n-1\}$, then property (2) implies that $X_{2^{k-1}}(\omega_i)^2 - X_{2^{k-1}}(\omega_i) = 0$ and $X_{2^k}(\omega_i) = 0$ for $i \in \{0, \ldots, 2^k - 1\}$. As X_{2^k} and $X_{2^{k-1}}^2 - X_{2^{k-1}}$ both have degree equal to 2^k, it follows that $X_{2^k} = (X_{2^{k-1}}^2 - X_{2^{k-1}})/\delta$ for some nonzero element $\delta \in \mathbb{F}$. Observing that $X_{2^k}(\beta_k) = X_{2^k}(\omega_{2^k}) = 1$ then shows that $\delta = X_{2^{k-1}}(\beta_k)^2 - X_{2^{k-1}}(\beta_k)$, completing the proof of property (3).

Property (4) follows from property (3) since it shows that X_1 is $\mathbb{F}_2$-linearised, and the recursive formula implies that if $X_{2^{k-1}}$ is $\mathbb{F}_2$-linearised for some $k \in \{1, \ldots, n-1\}$, then so too is X_{2^k}. Property (5) follows from property (4) since $\mathbb{F}$ has characteristic equal to two. □

3. Conversion between the Newton and LCH bases

The algorithms of this sections are the simplest of the paper, which we take advantage of to introduce several techniques that are applied again in later sections. The simplicity of the algorithms follows from the observation that the Newton basis polynomials exhibit a factorisation property which closely mimics that of the LCH basis described in property (1) of Lemma 2.1.

Lemma 3.1. *We have* $N_{2^k+i} = X_{2^k}(x) N_i(x + \beta_k)$ *for* $k \in \{0, \ldots, n-1\}$ *and* $i \in \{0, \ldots, 2^k - 1\}$.

Proof. Let $k \in \{0, \ldots, n-1\}$ and $i \in \{0, \ldots, 2^k - 1\}$. Then the definition of $\omega_0, \ldots, \omega_{2^n-1}$ implies that $\omega_{2^k+j} = \omega_j + \beta_k$ for $j \in \{0, \ldots, 2^k - 1\}$. Moreover, the definition of the Newton basis polynomials implies that $N_{2^k+i}(\omega_j) = 0$ for $j \in \{0, \ldots, 2^k + i - 1\}$, and $N_{2^k+i}(\omega_{2^k+i}) = 1$. Combining with property (2) of Lemma 2.1, it follows that $X_{2^k}(\omega_j) N_i(\omega_j + \beta_k) = 0 \times N_i(\omega_j + \beta_k) = 0$ for $j \in \{0, \ldots, 2^k - 1\}$, $X_{2^k}(\omega_{2^k+j}) N_i(\omega_{2^k+j} + \beta_k) = N_i(\omega_j) = 0$ for $j \in \{0, \ldots, i-1\}$, and $X_{2^k}(\omega_{2^k+i}) N_i(\omega_{2^k+i} + \beta_k) = N_i(\omega_i) = 1$. Thus, $N_{2^k+i}(x)$ and $X_{2^k}(x) N_i(x + \beta_k)$ are equal, since they agree on $2^k + i + 1$ distinct values and both have degree $2^k + i$. □

Roughly speaking, Lemma 3.1 and property (1) of Lemma 2.1 allow the quotient and remainder of polynomials in $\mathbb{F}[x]_{2^{k+1}}$ upon division by X_{2^k}, $k \in \{0, \ldots, n-1\}$, to be efficiently computed with respect to the Newton and LCH bases. As the quotient and remainder are unique and belong to $\mathbb{F}[x]_{2^k}$, equating their different representations on the two bases provides a means of reducing the conversion problem to shorter instances, suggesting that it may be solved efficiently by a

divide-and-conquer approach. However, the shift of variables that appears in the factorisations of the Newton basis polynomials only permits the quotients to be computed on the Newton basis with a shift of variables. As a result, we generalise the conversion problem itself to include a shift of variables in order to facilitate the development of divide-and-conquer algorithms.

Lemma 3.2. *Let $\ell \in \{2, \dots, 2^n\}$ and $k = \lceil \log_2 \ell \rceil - 1$. Then*

$$\sum_{i=0}^{\ell-1} f_i N_i(x+\lambda) = \sum_{i=0}^{\ell-1} h_i X_i(x) \tag{3.1}$$

for $f_0, \dots, f_{\ell-1}, \lambda, h_0, \dots, h_{\ell-1} \in \mathbb{F}$ if and only if

$$\sum_{i=0}^{2^k-1} f_i N_i(x+\lambda) = \sum_{i=0}^{\ell-2^k-1} (h_i + X_{2^k}(\lambda) h_{2^k+i}) X_i(x) + \sum_{i=\ell-2^k}^{2^k-1} h_i X_i(x) \tag{3.2}$$

and

$$\sum_{i=0}^{\ell-2^k-1} f_{2^k+i} N_i(x+\lambda+\beta_k) = \sum_{i=0}^{\ell-2^k-1} h_{2^k+i} X_i(x). \tag{3.3}$$

Proof. Let $\ell \in \{2, \dots, 2^n\}$ and $k = \lceil \log_2 \ell \rceil - 1$. Then $k \in \{0, \dots, n-1\}$ and $\ell - 2^k \leq 2^{k+1} - 2^k = 2^k$. Thus, for $i \in \{0, \dots, \ell - 2^k - 1\}$ and $\lambda \in \mathbb{F}$, Lemma 3.1 implies that $N_{2^k+i}(x+\lambda) = X_{2^k}(x+\lambda) N_i(x+\lambda+\beta_k)$, while properties (1) and (5) of Lemma 2.1 imply that

$$X_{2^k+i} = X_{2^k}(x+\lambda+\lambda) X_i(x) = X_{2^k}(x+\lambda) X_i(x) + X_{2^k}(\lambda) X_i(x).$$

It follows that for $f_0, \dots, f_{\ell-1}, \lambda, h_0, \dots, h_{\ell-1} \in \mathbb{F}$, the polynomials on either side of (3.2), which each have degree less than $\max(\ell - 2^k, 2^k) = 2^k$, are equal to the remainder upon division by $X_{2^k}(x+\lambda)$ of the polynomial on their respective side of (3.1). Similarly, the polynomials on either side of (3.3) are equal to the quotient upon division by $X_{2^k}(x+\lambda)$ of the polynomial on their respective side of (3.1). Uniqueness of the quotient and remainder therefore implies that (3.1) holds for $f_0, \dots, f_{\ell-1}, \lambda, h_0, \dots, h_{\ell-1} \in \mathbb{F}$ if and only if (3.2) and (3.3) hold. □

3.1. Conversion algorithms

Lemma 3.2 suggests recursive algorithms for converting polynomials in $\mathbb{F}[x]_\ell \subseteq \mathbb{F}[x]_{2^n}$ between the LCH basis and the basis of shifted Newton polynomials $\{N_i(x+\lambda) \mid i \in \{0, \dots, 2^n - 1\}\}$ for a given shift parameter $\lambda \in \mathbb{F}$. Given the coefficients f_i on the left-hand side of (3.1), recursive calls can be made on the polynomials (3.2) and (3.3) to compute their coefficients on the LCH basis. These coefficients can in turn be used to compute the coefficients h_i on the right-hand side of (3.1) by performing $\ell - 2^k$ additions, and $\ell - 2^k$ multiplications by $X_{2^k}(\lambda)$. For the inverse conversion, where we start with the coefficients h_i on the right-hand side of (3.1), performing the same additions and multiplications yields the coefficients of the polynomials (3.2) and (3.3) on the LCH basis. Then recursive calls can be made to obtain their coefficients on the shifted Newton bases $\{N_i(x+\lambda) \mid i \in \{0, \dots, 2^n - 1\}\}$ and $\{N_i(x+\lambda+\beta_k) \mid i \in \{0, \dots, 2^n - 1\}\}$, respectively, and thus the coefficients f_i on the left-hand side of (3.1).

To efficiently compute the elements $X_{2^k}(\lambda)$ by which we multiply during the algorithms, we take advantage of the property (5) of Lemma 2.1 and the observation that the initial shift parameter is only augmented for the recursive calls by the addition of entries of β. To this end,

we assume that $X_{2^i}(\beta_j)$ for $0 \le i < j < \lceil \log_2 \ell \rceil$ have been precomputed. The pseudocode for the conversion from a shifted Newton basis to the LCH basis is presented in Algorithm 1, while the pseudocode for the inverse conversion is presented in Algorithm 2. Each algorithm operates on a vector $(a_i)_{0 \le i < \ell}$ of field elements which initially contains the coefficients of a polynomial on the input basis, and has it entries overwritten by the algorithm with the coefficients of the polynomial on the output basis.

Algorithm 1 NewtonToLCH$(\ell, (X_{2^i}(\lambda))_{0 \le i < \lceil \log_2 \ell \rceil}, (a_i)_{0 \le i < \ell})$

Input: an integer $\ell \in \{1, \ldots, 2^n\}$; the vector $(X_{2^i}(\lambda))_{0 \le i < \lceil \log_2 \ell \rceil}$ for some $\lambda \in \mathbb{F}$; and the vector $(a_i)_{0 \le i < \ell}$ such that $a_i = f_i \in \mathbb{F}$ for $i \in \{0, \ldots, \ell - 1\}$.

Output: $a_i = h_i \in \mathbb{F}$ for $i \in \{0, \ldots, \ell - 1\}$ such that (3.1) holds.

1: **if** $\ell = 1$ **then return**
2: $k \leftarrow \lceil \log_2 \ell \rceil - 1$, $\ell' \leftarrow \ell - 2^k$, $k' \leftarrow \lceil \log_2 \ell' \rceil$
3: NewtonToLCH$(2^k, (X_{2^i}(\lambda))_{0 \le i < k}, (a_i)_{0 \le i < 2^k})$
4: NewtonToLCH$(\ell', (X_{2^i}(\lambda) + X_{2^i}(\beta_k))_{0 \le i < k'}, (a_{2^k + i})_{0 \le i < \ell'})$
5: **for** $i = 0, \ldots, \ell' - 1$ **do**
6: $\quad a_i \leftarrow a_i + X_{2^k}(\lambda) a_{2^k + i}$

Algorithm 2 LCHToNewton$(\ell, (X_{2^i}(\lambda))_{0 \le i < \lceil \log_2 \ell \rceil}, (a_i)_{0 \le i < \ell})$

Input: an integer $\ell \in \{1, \ldots, 2^n\}$; the vector $(X_{2^i}(\lambda))_{0 \le i < \lceil \log_2 \ell \rceil}$ for some $\lambda \in \mathbb{F}$; and the vector $(a_i)_{0 \le i < \ell}$ such that $a_i = h_i \in \mathbb{F}$ for $i \in \{0, \ldots, \ell - 1\}$.

Output: $a_i = f_i \in \mathbb{F}$ for $i \in \{0, \ldots, \ell - 1\}$ such that (3.1) holds.

1: **if** $\ell = 1$ **then return**
2: $k \leftarrow \lceil \log_2 \ell \rceil - 1$, $\ell' \leftarrow \ell - 2^k$, $k' \leftarrow \lceil \log_2 \ell' \rceil$
3: **for** $i = 0, \ldots, \ell' - 1$ **do**
4: $\quad a_i \leftarrow a_i + X_{2^k}(\lambda) a_{2^k + i}$
5: LCHToNewton$(2^k, (X_{2^i}(\lambda))_{0 \le i < k}, (a_i)_{0 \le i < 2^k})$
6: LCHToNewton$(\ell', (X_{2^i}(\lambda) + X_{2^i}(\beta_k))_{0 \le i < k'}, (a_{2^k + i})_{0 \le i < \ell'})$

Theorem 3.3. *Algorithms 1 and 2 are correct.*

Proof. We prove correctness for Algorithm 1 by induction on ℓ. The proof of correctness for Algorithm 2 uses similar arguments, and is omitted here. Alternatively, one may note that the transformation performed on the vector $(a_i)_{0 \le i < \ell}$ by the for-loop in Algorithm 1 is an involution. Algorithm 2 reverses the order of these transformations, thus performing the inverse transformation to Algorithm 1 overall.

Algorithm 1 is correct for all inputs with $\ell = 1$, since $X_0 = N_0 = 1$. Therefore, suppose that for some $\ell \in \{2, \ldots, 2^n\}$, the algorithm produces the correct output for all inputs with smaller values of ℓ. Moreover, suppose that the algorithm is given ℓ as an input, together with $(X_{2^i}(\lambda))_{0 \le i < \lceil \log_2 \ell \rceil}$ for some $\lambda \in \mathbb{F}$, and the vector $(a_i)_{0 \le i < \ell}$ with $a_i = f_i \in \mathbb{F}$ for $i \in \{0, \ldots, \ell - 1\}$. Let $k = \lceil \log_2 \ell \rceil - 1$, $\ell' = \ell - 2^k$ and $k' = \lceil \log_2 \ell' \rceil$, as computed in Line 2 of the algorithm, and $h_0, \ldots, h_{\ell - 1} \in \mathbb{F}$ be the unique elements that satisfy (3.1). Then Lemma 3.2 implies that (3.2) and (3.3) both hold. Therefore, as $2^k < \ell$, (3.2) and the induction hypothesis imply that Line 3 sets $a_i = h_i + h_{2^k + i} X_{2^k}(\lambda)$ for $i \in \{0, \ldots, \ell' - 1\}$, and $a_i = h_i$ for $i \in \{\ell', \ldots, 2^k - 1\}$. Property (5)

of Lemma 2.1 implies that $(X_{2^i}(\lambda) + X_{2^i}(\beta_k))_{0\le i<k'} = (X_{2^i}(\lambda+\beta_k))_{0\le i<k'}$. Consequently, as $\ell' < \ell$, (3.3) and the induction hypothesis imply that Line 4 sets $a_i = h_i$ for $i \in \{2^k, \dots, \ell-1\}$. It follows that Lines 5 and 6 set $a_i = (h_i + h_{2^k+i}X_{2^k}(\lambda)) + h_{2^k+i}X_{2^k}(\lambda) = h_i$ for $i \in \{0, \dots, \ell'-1\}$. Thus, the algorithm terminates with $a_i = h_i$ for $i \in \{0, \dots, \ell-1\}$, as required. □

3.2. Complexity

When implementing Algorithms 1 and 2, subvectors of the input vector $(a_i)_{0\le i<\ell}$ may be represented by a parameter that indicates the offset of their first entry, rather than by replicating the subvector in memory. Moreover, if the vectors $(X_{2^i}(\lambda))_{0\le i<\lceil\log_2 \ell\rceil}$ and $(X_{2^i}(\lambda)+X_{2^i}(\beta_k))_{0\le i<k'}$ are always cleared from memory when the algorithm returns, then storing the vectors and those of any subsequent recursive calls requires only $O(\log^2 \ell)$ fields elements to be stored in auxiliary space at all times. It follows that Algorithms 1 and 2 can be implemented so that only $O(\log^2 \ell)$ field elements are required to be stored in auxiliary space, which includes the storage of precomputed elements.

The recurrence relation of property (3) of Lemma 2.1 allows $X_{2^i}(\beta_j)$ for $0 \le i < j < \lceil\log_2 \ell\rceil$ to be computed with $O(\log^2 \ell)$ operations in $\mathbb{F}$ (see Remark 5.6). Subsequently, the recurrence relation can again be used to compute $(X_{2^i}(\lambda))_{0\le i<\lceil\log_2 \ell\rceil}$ for a desired $\lambda \in \mathbb{F}$ with a further $O(\log \ell)$ operations. If $\lambda = 0$, then the vector simply contains all zeros. It follows that all precomputations for Algorithms 1 and 2 can be performed with $O(\log^2 \ell)$ operations in $\mathbb{F}$. The following theorem bounds the number of operations then performed by the algorithms themselves.

Theorem 3.4. *Algorithms 1 and 2 perform at most* $(\lfloor \ell/2 \rfloor - 1)\lceil\log_2 \ell\rceil + \ell - 1$ *additions in* $\mathbb{F}$*, and at most* $\lfloor \ell/2 \rfloor\lceil\log_2 \ell\rceil$ *multiplications in* $\mathbb{F}$.

Proof. We prove the bounds for Algorithm 1 only, since it is clear that the two algorithms perform the same number of operations when given identical values of ℓ. If $\ell = 1$, then Algorithm 1 performs no additions or multiplications, matching the bounds of the theorem. Proceeding by induction, suppose that the algorithm is called with $\ell \in \{2, \dots, 2^n\}$, and that the two bounds hold for all smaller values of ℓ. Let $k = \lceil\log_2 \ell\rceil - 1$, $\ell' = \ell - 2^k$ and $k' = \lceil\log_2 \ell'\rceil$, as computed in Line 2 of the algorithm. Then, as $2^k < \ell$, the induction hypothesis implies that Line 3 performs at most $(2^{k-1}-1)k + 2^k - 1$ additions and at most $2^{k-1}k$ multiplications. The computation of the vector $(X_{2^i}(\lambda) + X_{2^i}(\beta_k))_{0\le i<k'}$ in Line 4 requires k' additions, where $k' \le k$ since $\ell' = \ell - 2^k \le 2^{k+1} - 2^k = 2^k$. Thus, as $\ell' < \ell$, the induction hypothesis implies that Line 4 performs at most $\lfloor \ell'/2 \rfloor k' + \ell' - 1 \le (\lfloor \ell/2 \rfloor - 2^{k-1})k + \ell - 2^k - 1$ additions and at most $\lfloor \ell'/2 \rfloor k' \le (\lfloor \ell/2 \rfloor - 2^{k-1})k$ multiplications. Lines 5 and 6 perform $\ell' = \ell - \lceil 2^{k+1}/2 \rceil \le \ell - \lceil \ell/2 \rceil = \lfloor \ell/2 \rfloor$ additions and multiplications. Summing these bounds, it follows that the algorithms performs at most $(\lfloor \ell/2 \rfloor - 1)(k+1) + \ell - 1 = (\lfloor \ell/2 \rfloor - 1)\lceil\log_2 \ell\rceil + \ell - 1$ additions, and at most $\lfloor \ell/2 \rfloor(k+1) = \lfloor \ell/2 \rfloor\lceil\log_2 \ell\rceil$ multiplications. □

3.3. Conversion between the standard Newton basis and the LCH basis

For $i \in \{0, \dots, 2^n - 1\}$, define polynomials

$$\bar{X}_i = \prod_{k=0}^{n-1} \prod_{j=0}^{2^k[i]_k - 1} (x - \omega_j) \quad \text{and} \quad \bar{N}_i = \prod_{j=0}^{i-1} (x - \omega_j).$$

Then $\{\bar{N}_0, \dots, \bar{N}_{2^n-1}\}$ is what is usually defined to be the Newton basis associated with the points $\omega_0, \dots, \omega_{2^n-1}$. Consequently, we refer to the basis as the standard Newton basis, and hereafter

refer to $\{N_0, \ldots, N_{2^n-1}\}$ as the scaled Newton basis. The two bases coincide when β is a Cantor basis, since $X_{2^0}, \ldots, X_{2^{n-1}} \in \mathbb{F}_2[x]$ (see Gao and Mateer, 2010, Appendix A), and, thus, Lemma 3.1 implies that the polynomials of the scaled Newton basis are all monic. However, the bases do not coincide in general.

The polynomials $\bar{X}_i$ and $\bar{N}_i$ share many of the properties of their respective scalar multiples X_i and N_i. In particular, $\bar{X}_{2^0}, \ldots, \bar{X}_{2^{n-1}}$ are $\mathbb{F}_2$-linearised, and Lemma 3.2 still holds if all X_i's and N_i's are replaced by their counterparts $\bar{X}_i$ and $\bar{N}_i$. It follows that Algorithms 1 and 2 can be used to convert between the standard Newton basis and the basis $\{\bar{X}_0, \ldots, \bar{X}_{2^n-1}\}$ by similarly replacing X_i's by $\bar{X}_i$'s in the algorithms. Property (3) of Lemma 2.1 implies that $\bar{X}_{2^0} = x$ and $\bar{X}_{2^k} = \bar{X}_{2^{k-1}}^2 - \bar{X}_{2^{k-1}}(\beta_{k-1})\bar{X}_{2^{k-1}}$ for $k \in \{1, \ldots, n-1\}$ (see also von zur Gathen and Gerhard, 1996, Section 2). Thus, the precomputations for the modified algorithms can once again be performed with $O(\log^2 \ell)$ operations in $\mathbb{F}$.

Let $X_i = \bar{X}_i/\Delta_i$ for $i \in \{0, \ldots, 2^n - 1\}$. Then $\Delta_i = \prod_{k=0}^{n-1} \bar{X}_{2^k}(\beta_k)^{[i]_k}$ for $i \in \{0, \ldots, 2^n - 1\}$. Thus, for $k \in \mathbb{N}$, the elements of the set $\{\Delta_0, \ldots, \Delta_{2^k-1}\}$ can be computed with $O(2^k)$ operations in $\mathbb{F}$ by traversing the set using the k-bit binary reflected Gray code (see Bitner et al., 1976; Knuth, 2005), so that first computed element is $\Delta_0 = 1$, and each successive element can be computed by multiplying or dividing the previous element by one of $\bar{X}_{2^0}(\beta_0), \ldots, \bar{X}_{2^{n-1}}(\beta_{n-1})$. It follows that conversion between $\{\bar{X}_0, \ldots, \bar{X}_{2^n-1}\}$ and the LCH basis for polynomials in $\mathbb{F}[x]_\ell$ can be performed with $O(\ell)$ operations in $\mathbb{F}$. Therefore, it possible to convert polynomials in $\mathbb{F}[x]_\ell$ between the standard Newton basis and the LCH basis with only $O(\ell)$ additional operations in $\mathbb{F}$ than required by the algorithms of Section 3.1 for conversion between the scaled Newton basis and the LCH basis.

4. Truncated evaluation and interpolation on the LCH basis

In this section, we consider evaluation and interpolation with respect to the LCH basis for polynomials in $\mathbb{F}[x]_\ell$ and evaluations points of the form $\omega_0 + \lambda, \ldots, \omega_{c-1} + \lambda$ for $\ell, c \in \{1, \ldots, 2^n\}$ and $\lambda \in \mathbb{F}$, where $c = \ell$ in the case of interpolation. We build upon the work of Lin, Chung and Han (2014) who propose quasi-linear time algorithms for the case $\ell = c = 2^{k+1}$ for some $k \in \{0, \ldots, n-1\}$. However, we do not reduce solving the general problems to applications of their algorithms by embedding into instances with such parameters, e.g., by zero padding. Instead, we provide "truncated" algorithms in the style of van der Hoeven's algorithms for multiplicative FFTs (van der Hoeven, 2004). Moreover, we combine techniques developed in Section 3 with additional novel methods to ensure a polylogarithmic bound on the number of field elements that are required to be stored in auxiliary space by the algorithms.

4.1. Reductions

Consider a polynomial $f = \sum_{i=0}^{2^{k+1}-1} h_i X_i$ with $k \in \{0, \ldots, n-1\}$ and $h_0, \ldots, h_{2^{k+1}-1} \in \mathbb{F}$. Then properties (1) and (2) of Lemma 2.1 imply that $X_{2^k+i}(\omega_j) = 0$ and $X_{2^k+i}(\omega_{2^k+j}) = X_i(\omega_{2^k+j})$ for $i, j \in \{0, \ldots, 2^k - 1\}$. Thus, the polynomials

$$f_0 = \sum_{i=0}^{2^k-1} h_i X_i \quad \text{and} \quad f_1 = \sum_{i=0}^{2^k-1} (h_i + h_{2^k+i})\, X_i$$

satisfy $f(\omega_j) = f_0(\omega_j)$ and $f(\omega_{2^k+j}) = f_1(\omega_{2^k+j})$ for $j \in \{0, \ldots, 2^k - 1\}$. Given the coefficients of f on the LCH basis, only 2^k additions are needed to compute those of f_0 and f_1, and vice

versa. Thus, the problem of evaluating f at the points $\omega_0, \ldots, \omega_{2^{k+1}-1}$ can be efficiently reduced to evaluating f_0 at the points $\omega_0, \ldots, \omega_{2^k-1}$, and f_1 at the points $\omega_{2^k}, \ldots, \omega_{2^{k+1}-1}$. Similarly, the problem of recovering the coefficients of f on the LCH basis given $f(\omega_0), \ldots, f(\omega_{2^{k+1}-1})$ can be efficiently reduced to that of recovering the coefficients of f_0 and f_1 given $f_0(\omega_0), \ldots, f_0(\omega_{2^k-1})$ and $f_1(\omega_{2^k}), \ldots, f_1(\omega_{2^{k+1}-1})$.

Both $\{\omega_0, \ldots, \omega_{2^{k+1}-1}\}$ and $\{\omega_0, \ldots, \omega_{2^k-1}\}$ are linear subspaces of $\mathbb{F}$, while $\{\omega_{2^k}, \ldots, \omega_{2^{k+1}-1}\}$ is only an affine subspace, since $\omega_{2^k+i} = \omega_i + \beta_k$ for $i \in \{0, \ldots, 2^k - 1\}$. This observation leads Lin, Chung and Han to instead consider evaluation points of the form $\omega_0 + \lambda, \ldots, \omega_{2^{k+1}-1} + \lambda$ for some $\lambda \in \mathbb{F}$, so as to facilitate the development of divide-and-conquer algorithms. The following lemma, for which the case $\ell = 2^{k+1}$ is due to Lin, Chung and Han, and which provides the foundations of our algorithms, then generalises the above reduction accordingly.

Lemma 4.1. *Let $\ell \in \{1, \ldots, 2^n\}$, $f = \sum_{i=0}^{\ell-1} h_i X_i$ with $h_0, \ldots, h_{\ell-1} \in \mathbb{F}$, and $\lambda \in \mathbb{F}$. For some $k \in \{0, \ldots, n-1\}$ such that $k \geq \lceil \log_2 \ell \rceil - 1$, define $\ell' = \min(\ell, 2^k)$, $\ell'' = \ell - \ell'$ and*

$$f_t = \sum_{i=0}^{\ell''-1} \left(h_i + (t + X_{2^k}(\lambda))\, h_{2^k+i}\right) X_i + \sum_{i=\ell''}^{\ell'-1} h_i X_i \quad \text{for } t \in \{0, 1\}. \tag{4.1}$$

Then $f(\omega_{2^k t+s} + \lambda) = f_t(\omega_s + \lambda + t\beta_k)$ for $t \in \{0, 1\}$ and $s \in \{0, \ldots, 2^k - 1\}$.

Proof. Let $\ell \in \{1, \ldots, 2^n\}$, $\lambda \in \mathbb{F}$, $k \in \{0, \ldots, n-1\}$ such that $k \geq \lceil \log_2 \ell \rceil - 1$, $\ell' = \min(\ell, 2^k)$ and $\ell'' = \ell - \ell'$. Then it follows from the definition of $\omega_0, \ldots, \omega_{2^n-1}$ that $X_i(\omega_{2^k t+s} + \lambda) = X_i(\omega_s + \lambda + t\beta_k)$ for $i \in \{0, \ldots, \ell' - 1\}$, $t \in \{0, 1\}$ and $s \in \{0, \ldots, 2^k - 1\}$. The choice of k implies that $\ell'' = \max(\ell - 2^k, 0) \leq 2^k$. Thus, properties (1), (2) and (5) of Lemma 2.1 imply that

$$\begin{aligned} X_{2^k+i}(\omega_{2^k t+s} + \lambda) &= X_{2^k}(\omega_{2^k t+s} + \lambda)\, X_i(\omega_{2^k t+s} + \lambda) \\ &= (X_{2^k}(\omega_{2^k t+s}) + X_{2^k}(\lambda))\, X_i(\omega_s + \lambda + t\beta_k) \\ &= (t + X_{2^k}(\lambda))\, X_i(\omega_s + \lambda + t\beta_k) \end{aligned}$$

for $i \in \{0, \ldots, \ell'' - 1\}$, $t \in \{0, 1\}$ and $s \in \{0, \ldots, 2^k - 1\}$. Therefore, the lemma holds if $f = X_i$ for some $i \in \{0, \ldots, \ell' - 1\} \cup \{2^k, \ldots, 2^k + \ell'' - 1\} = \{0, \ldots, \ell - 1\}$, and, thus, for all f of the form $\sum_{i=0}^{\ell-1} h_i X_i$ with $h_0, \ldots, h_{\ell-1} \in \mathbb{F}$ by linearity. □

4.2. Truncated evaluation on the LCH basis

Given the coefficients of a polynomial $f \in \mathbb{F}[x]_\ell \subseteq \mathbb{F}[x]_{2^n}$ on the LCH basis, Lemma 4.1 suggests a recursive algorithm for evaluating the polynomial at $c \in \{1, \ldots, 2^n\}$ evaluation points of the form $\omega_0 + \lambda, \ldots, \omega_{c-1} + \lambda$ for some $\lambda \in \mathbb{F}$. If $\ell = c = 1$, then the algorithm only has to return the provided coefficient since f is a constant polynomial. For the remaining cases, after letting $k = \lceil \log_2 \max(\ell, c) \rceil - 1$ and taking f_0 and f_1 to be the polynomials defined in (4.1), Lemma 4.1 then reduces the problem to recursively evaluating f_0 at the points $\omega_0 + \lambda, \ldots, \omega_{\min(c,2^k)-1} + \lambda$ and, if $c > 2^k$, evaluating f_1 at the points $\omega_0 + (\lambda + \beta_k), \ldots, \omega_{c-2^k-1} + (\lambda + \beta_k)$. The coefficients of f_0 on the LCH basis can be computed with $\ell'' = \max(\ell - 2^k, 0)$ additions and ℓ'' multiplications by $X_{2^k}(\lambda)$. Then, if needed, (4.1) implies that the coefficients of f_1 can be computed with a further ℓ'' additions.

Simultaneously storing the coefficients of f_0 and f_1 naively requires space for $2\ell'$ field elements. However, if $\ell \geq 2^{\lceil \log_2 \max(\ell,c) \rceil - 1}$, then $2\ell' = 2^{\lceil \log_2 \max(\ell,c) \rceil}$ is potentially almost double the length $\max(\ell, c)$ of the evaluation problem, preventing a polylogarithmic bound on auxiliary

space. This problem does not arise for parameters $\ell = c = 2^{k+1}$ for some $k \in \{0, \ldots, n-1\}$, as considered by Lin, Chung and Han, since it then holds that $2\ell' = \max(\ell, c)$. To address the problem, we observe that (4.1) implies that f_0 and f_1 have $\ell' - \ell''$ coefficients in common, which we use to store their coefficients with only $\ell' + \ell'' = \ell$ field elements. However, we then have to ensure that the common coefficients are available to both recursive evaluations. That is, we must ensure that they are preserved by the first of two recursive evaluations. We enforce this requirement by augmenting the evaluation problem itself to additionally require that the coefficients of $X_c, \ldots, X_{\ell-1}$ in f are returned alongside the evaluations of the polynomial if $c < \ell$.

The pseudocode for our algorithm that solves the augmented evaluation problem is presented in Algorithm 3. Similar to the algorithms of Section 3, the algorithm is presented under the assumption that $X_{2^i}(\beta_j)$ for $0 \le i < j < \lceil \log_2 \max(\ell, c) \rceil$ have been precomputed for the input values of ℓ and c. If $\ell < c$, then the input requirements of the algorithm do not specify the initial value of a_i for $i \in \{\ell, \ldots, c-1\}$. These entries are overwritten by Lines 8 and 9 over the course of the algorithm, so may be initially filled with any combination of field elements, e.g., zeros. In Line 11 of the algorithm, $(a_{2^k+i})_{0 \le i < t''} \,\|\, (a_i)_{t'' \le i < t'}$ denotes the concatenation of the vectors $(a_{2^k+i})_{0 \le i < t''}$ and $(a_i)_{t'' \le i < t'}$, which is taken to be $(a_{2^k+i})_{0 \le i < t''}$ if $t' = t''$.

Algorithm 3 $\mathsf{LCHEval}(\ell, c, (X_{2^i}(\lambda))_{0 \le i < \lceil \log_2 \ell \rceil}, (a_i)_{0 \le i < \max(\ell, c)})$

Input: integers $\ell, c \in \{1, \ldots, 2^n\}$; the vector $(X_{2^i}(\lambda))_{0 \le i < \lceil \log_2 \ell \rceil}$ for some $\lambda \in \mathbb{F}$; and the vector $(a_i)_{0 \le i < \max(\ell, c)}$ such that $a_i = h_i \in \mathbb{F}$ for $i \in \{0, \ldots, \ell - 1\}$.

Output: $a_i = f(\omega_i + \lambda)$ for $i \in \{0, \ldots, c-1\}$, where $f = \sum_{i=0}^{\ell-1} h_i X_i$; and, if $c < \ell$, $a_i = h_i$ for $i \in \{c, \ldots, \ell - 1\}$.

1: **if** $\ell = 1$ **and** $c = 1$ **then return**
2: $k \leftarrow \lceil \log_2 \max(\ell, c) \rceil - 1$, $\ell' \leftarrow \min(\ell, 2^k)$, $\ell'' \leftarrow \ell - \ell'$, $c_0 \leftarrow \min(c, 2^k)$, $c_1 \leftarrow c - c_0$
3: **for** $i = 0, \ldots, \ell'' - 1$ **do**
4: $\quad a_i \leftarrow a_i + X_{2^k}(\lambda) a_{2^k+i}$
5: **if** $c_1 > 0$ **then**
6: $\quad$ **for** $i = 0, \ldots, \ell'' - 1$ **do**
7: $\quad\quad a_{2^k+i} \leftarrow a_i + a_{2^k+i}$
8: $\quad$ **for** $i = \ell'', \ldots, \min(\ell', c_1) - 1$ **do**
9: $\quad\quad a_{2^k+i} \leftarrow a_i$
10: $\quad t' \leftarrow \max(\ell', c_1)$, $t'' \leftarrow \max(\ell'', c_1)$
11: $\quad \mathsf{LCHEval}(\ell', c_1, (X_{2^i}(\lambda) + X_{2^i}(\beta_k))_{0 \le i < \lceil \log_2 \ell' \rceil}, (a_{2^k+i})_{0 \le i < t''} \,\|\, (a_i)_{t'' \le i < t'})$
12: $\quad$ **for** $i = c_1, \ldots, \ell'' - 1$ **do**
13: $\quad\quad a_{2^k+i} \leftarrow a_i + a_{2^k+i}$
14: $\mathsf{LCHEval}(\ell', c_0, (X_{2^i}(\lambda))_{0 \le i < \lceil \log_2 \ell' \rceil}, (a_i)_{0 \le i < 2^k})$
15: **for** $i = c_0, \ldots, \ell'' - 1$ **do**
16: $\quad a_i \leftarrow a_i + X_{2^k}(\lambda) a_{2^k+i}$

Theorem 4.2. *Algorithm 3 is correct.*

Proof. We prove the theorem by induction on $\lceil \log_2 \max(\ell, c) \rceil$. If $\lceil \log_2 \max(\ell, c) \rceil = 0$, then $\ell = c = 1$ and Algorithm 3 produces the correct output since $f = h_0$ is a constant polynomial. Therefore, for some $k \in \{0, \ldots, n-1\}$, suppose that the algorithm produces the correct output for all inputs with $\lceil \log_2 \max(\ell, c) \rceil \le k$. Let $\ell, c \in \{1, \ldots, 2^n\}$ such that $\lceil \log_2 \max(\ell, c) \rceil = k + 1$ and $h_0, \ldots, h_{\ell-1} \in \mathbb{F}$. Suppose that the algorithm is called with ℓ and c as inputs, together with

$(X_{2^i}(\lambda))_{0\le i<\lceil \log_2 \ell\rceil}$ for some $\lambda \in \mathbb{F}$, and $a_i = h_i$ for $i \in \{0, \dots, \ell-1\}$. Define integers k, ℓ', ℓ'', c_0 and c_1 as in Line 2 of the algorithm. Finally, define $f = \sum_{i=0}^{\ell-1} h_i X_i$ as in the output requirements of the algorithm, and $f_0, f_1 \in \mathbb{F}[x]_{\ell'}$ by (4.1).

We have $\ell' \le 2^k$ and $\{0, \dots, \ell'-1\} \cup \{2^k, \dots, 2^k + \ell'' - 1\} = \{0, \dots, \ell-1\}$. Thus, after Lines 3 and 4 have been performed, the entries of the subvector $(a_i)_{0\le i<2^k}$ satisfy

$$a_i = \begin{cases} h_i + X_{2^k}(\lambda) h_{2^k+i} & \text{if } i < \ell'', \\ h_i & \text{if } \ell'' \le i < \ell', \\ * & \text{otherwise,} \end{cases} \tag{4.2}$$

where an asterisks denotes an entry that is unspecified by the input requirements of the algorithm and has so far not been overwritten. In particular, comparing with (4.1) shows that Lines 3 and 4 fill the subvector $(a_i)_{0\le i<\ell'}$ with the coefficients of f_0 on the LCH basis.

We assume to begin with that $c_1 > 0$. Define $t' = \max(\ell', c_1)$ and $t'' = \max(\ell'', c_1)$ as in Line 10 of the algorithm. Then $2^k + t'' = \max(\ell, c)$ and $t'' \le t' \le \max(\ell, c)$, since $\ell'' \le \ell' \le \ell$ and $c_1 \le c$. Thus, $(a_{2^k+i})_{0\le i<t''}$ and $(a_i)_{t''\le i<t'}$ are indeed subvectors of $(a_i)_{0\le i<\max(\ell,c)}$, and their concatenation $(a_{2^k+i})_{0\le i<t''} \,\|\, (a_i)_{t''\le i<t'}$ has length $t' = \max(\ell', c_1)$, as required to be consistent with the remaining inputs of the recursive call of Line 11.

The entries of the subvector $(a_{2^k+i})_{0\le i<t''}$ are unchanged by Lines 3 and 4 of the algorithm. Thus, $a_{2^k+i} = h_{2^k+i}$ for $i \in \{0, \dots, \ell''-1\}$, as on input, when the for-loop commences in Line 6. Therefore, (4.2) implies that after Lines 6 to 9 have been performed, the entries of the subvector $(a_{2^k+i})_{0\le i<t''}$ satisfy

$$a_{2^k+i} = \begin{cases} h_i + (1 + X_{2^k}(\lambda)) h_{2^k+i} & \text{if } i < \ell'', \\ h_i & \text{if } \ell'' \le i < \min(\ell', c_1), \\ * & \text{otherwise.} \end{cases} \tag{4.3}$$

That is, Lines 6 to 9 fill the subvector with as many coefficients of f_1 on the LCH basis as possible.

If $t' \ne \ell'$, then $c_1 > \ell' \ge \ell''$ and, thus, $t' = t''$. It follows in this case that $\{\ell'', \dots, \min(\ell', c_1) - 1\}$ and $\{t'', \dots, t'-1\}$ are disjoint and their union is $\{\ell'', \dots, \ell'-1\}$. If $t' = \ell'$, then

$$\{\ell'', \dots, \min(\ell', c_1) - 1\} = \{\ell'', \dots, c_1 - 1\} = \{\ell'', \dots, t'' - 1\}$$

and $\{t'', \dots, t'-1\} = \{t'', \dots, \ell'-1\}$ are once again disjoint and have union equal to $\{\ell'', \dots, \ell'-1\}$. Therefore, if we let $(b_i)_{0\le i<t'} = (a_{2^k+i})_{0\le i<t''} \,\|\, (a_i)_{t''\le i<t'}$ be the vector that is passed to the recursive call of Line 11, then (4.2) and (4.3) imply that its entries satisfy

$$b_i = \begin{cases} h_i + (1 + X_{2^k}(\lambda)) h_{2^k+i} & \text{if } i < \ell'', \\ h_i & \text{if } \ell'' \le i < \ell', \\ * & \text{otherwise.} \end{cases}$$

It follows that the subvector $(b_i)_{0\le i<\ell'}$ contains the coefficients of f_1 on the LCH basis. Property (5) of Lemma 2.1 implies that the third input passed to the recursive call of Line 11 is $(X_{2^i}(\lambda) + X_{2^i}(\beta_k))_{0\le i<\lceil \log_2 \ell'\rceil} = (X_{2^i}(\lambda + \beta_k))_{0\le i<\lceil \log_2 \ell'\rceil}$. Thus, as $\lceil \log_2 \max(\ell', c_1)\rceil \le k$, the induction hypothesis implies that after Line 11 has been performed, the entries of $(b_i)_{0\le i<t'}$ satisfy

$$b_i = \begin{cases} f_1(\omega_i + \lambda + \beta_k) & \text{if } i < c_1, \\ h_i + (1 + X_{2^k}(\lambda)) h_{2^k+i} & \text{if } c_1 \le i < \ell'', \\ h_i & \text{otherwise.} \end{cases}$$

Consequently, Lemma 4.1 implies that Line 11 sets $a_{2^k+i} = f(\omega_{2^k+i} + \lambda)$ for $i \in \{0, \dots, c_1 - 1\}$. Moreover, the subvector $(a_i)_{t'' \le i < t'}$ is unchanged by Line 11, since $\{t'', \dots, t'-1\} \subseteq \{\ell'', \dots, \ell'-1\}$. Thus, the entries of $(a_i)_{0 \le i < 2^k}$ still satisfy (4.2) when the for-loop commences in Line 12. It follows that Lines 12 and 13 set $a_{2^k+i} = (h_i + X_{2^k}(\lambda)h_{2^k+i}) + (h_i + (1 + X_{2^k}(\lambda))h_{2^k+i}) = h_{2^k+i}$ for $i \in \{c_1, \dots, \ell'' - 1\}$, restoring the entries to their initial values. Therefore, if $c_1 > 0$, then after Lines 5 to 13 have been performed, the entries of the subvector $(a_i)_{0 \le i < 2^k}$ satisfy (4.2), and the entries of the subvector $(a_{2^k+i})_{0 \le i < t''}$ satisfy

$$a_{2^k+i} = \begin{cases} f(\omega_{2^k+i} + \lambda) & \text{if } i < c_1, \\ h_i & \text{otherwise.} \end{cases} \tag{4.4}$$

This statement also holds if $c_1 = 0$. Indeed, Lines 5 to 13 have no effect in this case, and a_{2^k+i} is initially equal to h_{2^k+i} for $i \in \{0, \dots, t'' - 1\}$, since $2^k + t'' = 2^k + \ell'' = \ell$.

We have $\lceil \log_2 \max(\ell', c_0) \rceil = k$, since the definition of k implies that $\ell \ge 2^k$ or $c \ge 2^k$. Thus, the subvector $(a_i)_{0 \le i < 2^k}$ has length that is consistent with the inputs ℓ' and c_0 of the recursive call of Line 14. Moreover, as (4.2) holds for $i \in \{0, \dots, 2^k - 1\}$, the induction hypothesis and Lemma 4.1 imply that after recursive call has been performed, the entries of the subvector $(a_i)_{0 \le i < 2^k}$ satisfy

$$a_i = \begin{cases} f(\omega_i + \lambda) & \text{if } i < c_0, \\ h_i + X_{2^k}(\lambda)h_{2^k+i} & \text{if } c_0 \le i < \ell'', \\ h_i & \text{otherwise.} \end{cases}$$

As $c_1 \le c_0$, $\ell'' \le t''$ and (4.4) holds for $i \in \{0, \dots, t'' - 1\}$, Lines 15 and 16 then set $a_i = (h_i + X_{2^k}(\lambda)h_{2^k+i}) + X_{2^k}(\lambda)h_{2^k+i} = h_i$ for $i \in \{c_0, \dots, \ell'' - 1\}$, restoring the entries to their initial values. Hence, the algorithm terminates with $a_i = f(\omega_i + \lambda)$ for $i \in \{0, \dots, c_0 - 1\} \cup \{2^k, \dots, 2^k + c_1 - 1\} = \{0, \dots, c-1\}$, and $a_i = h_i$ for $i \in \{c_0, \dots, 2^k - 1\} \cup \{2^k + c_1, \dots, 2^k + t'' - 1\} = \{c_0, \dots, \max(\ell, c) - 1\}$, as required. □

In Line 11 of Algorithm 3, the vector $(a_{2^k+i})_{0 \le i < t''} \,\|\, (a_i)_{t'' \le i < t'}$ can be obtained as the subvector formed by the last t' entries of the cyclic left shift by $t' - t''$ positions of $(a_i)_{t'' \le i < \max(\ell, c)}$. Thus, Line 11 of the algorithm can be realised by first cyclically shifting $(a_i)_{t'' \le i < \max(\ell, c)}$ left by $t' - t''$ positions, then passing its last t' entries to the recursive call, and finally cyclically shifting the vector right by $t' - t''$ positions. Using in-place algorithms (see Gries and Mills, 1981; Shene, 1997; Furia, 2014) the cyclic shifts can be performed with only $O(1)$ additional field elements stored in auxiliary space, and, since the vector has length $\max(\ell, c) - t'' = 2^k$, $O(2^k)$ movements of its entries, where a movement involves either assigning a value into an entry of the vector or copying one of its entries elsewhere. It follows that $O(\max(\ell, c))$ movements are performed overall, since the difference $t' - t''$ for the initial call of the algorithm and each of its subsequent recursive calls is always zero if the initial parameters satisfy $\max(\ell, c) = 2^{\lceil \log_2 \max(\ell, c) \rceil}$, nonzero at most once if the parameters satisfy $c \le \ell$, and nonzero at most twice otherwise. We note that this bound also holds for the entire algorithm, since Line 9 is performed $\max(c - \ell, 0)$ times over the course of the algorithm.

Combining the method of cyclic shifts with techniques described in Section 3, such as representing subvectors by an offset variable, Algorithm 3 can be implemented so that only $O(\log^2 \max(\ell, c))$ field elements are required to be stored auxiliary space. Similarly, all precomputations for the algorithm can be performed with $O(\log^2 \max(\ell, c))$ operations in $\mathbb{F}$. The number of field operations performed by the algorithm itself is bounded by the following theorem.

Theorem 4.3. *Algorithm 3 performs at most* $(c-1)(\lceil \log_2 \min(\ell, c)\rceil + 1) + 2(\ell - 1)$ *additions in* $\mathbb{F}$*, and at most* $(c-1)\lceil \log_2 \min(\ell, c/2)\rceil/2 + 2(\ell-1)$ *multiplications in* $\mathbb{F}$*.*

We prove Theorem 4.3 with the aid of the following lemma.

Lemma 4.4. *If* $\ell \leq 2^r$ *and* $c = 2^r$ *for some* $r \in \{0, \ldots, n\}$*, then Algorithm 3 performs at most* $(2^r - 1)(\lceil \log_2 \ell'\rceil + 1)$ *additions in* $\mathbb{F}$*, and at most* $2^{r-1}\lceil \log_2 \ell'\rceil$ *multiplications in* $\mathbb{F}$*.*

Proof. We prove the lemma by induction on r. If $r = 0$, then $\ell = c = 1$ and Algorithm 3 performs no operations in $\mathbb{F}$, matching the bounds of the lemma. Therefore, suppose that the bounds of the lemma hold for some $r \in \{0, \ldots, n-1\}$. Then, when Algorithm 3 is called with parameters $\ell \leq 2^{r+1}$ and $c = 2^{r+1}$, Line 2 sets $k = r$, $\ell' = \min(\ell, 2^r)$, $\ell'' = \max(\ell - 2^r, 0)$ and $c_0 = c_1 = 2^r > 0$. It follows that Lines 3 to 10 perform $2\ell''$ additions and ℓ'' multiplications. The induction hypothesis implies that Lines 11 and 14 perform at most $2(2^r-1)(\lceil \log \ell'\rceil + 1) + \lceil \log_2 \ell'\rceil$ additions, and at most $2^r\lceil \log_2 \ell'\rceil$ multiplications. Finally, Lines 12, 13, 15 and 16 perform no operations in $\mathbb{F}$, since $\ell'' \leq 2^r = c_i$ for $i \in \{0, 1\}$. Summing these bounds, it follows that the algorithm performs at most $(2^{r+1}-1)(\lceil \log_2 \ell'\rceil + 1) + 2\ell'' - 1$ additions, and at most $2^r\lceil \log_2 \ell'\rceil + \ell''$ multiplications. If $\ell \leq 2^r$, then $\ell' = \ell$ and $\ell'' = 0$. If $\ell > 2^r$, then $\lceil \log_2 \ell'\rceil = r = \lceil \log_2 \ell\rceil - 1$ and $\ell'' \leq 2^r$. In either case, we find that the algorithm performs at most $(2^{r+1} - 1)(\lceil \log_2 \ell\rceil + 1)$ additions, and at most $2^r\lceil \log_2 \ell\rceil$ multiplications. □

Proof of Theorem 4.3. We prove the theorem by induction on $\lceil \log_2 \max(\ell, c)\rceil$. If $\lceil \log_2 \max(\ell, c)\rceil$ is equal to zero, then $\ell = c = 1$ and Algorithm 3 performs no operations in $\mathbb{F}$, matching the bounds of the theorem. Therefore, for some $k \in \{0, \ldots, n-1\}$, suppose that the bounds stated in the theorem hold for all inputs with $\lceil \log_2 \max(\ell, c)\rceil \leq k$. Moreover, suppose that the algorithm is called with inputs $\ell, c \in \{1, \ldots, 2^n\}$ such that $\lceil \log_2 \max(\ell, c)\rceil = k + 1$. Let $\ell' = \min(\ell, 2^k)$, $\ell'' = \ell - \ell'$, $c_0 = \min(c, 2^k)$ and $c_1 = c - c_0$, as computed in Line 2 of the algorithm.

Suppose to begin with that $c_1 = 0$. Then $c \leq 2^k < \ell$, $\ell' = 2^k$, $c_0 = c$, $\lceil \log_2 \min(\ell', c_0)\rceil = \lceil \log_2 \min(\ell, c)\rceil$ and $\lceil \log_2 \min(\ell', c_0/2)\rceil = \lceil \log_2 \min(\ell, c/2)\rceil$. Thus, Lines 3 and 4 perform $\ell'' = \ell - 2^k$ additions and multiplications, while Lines 5 to 13 perform no operations in $\mathbb{F}$. Moreover, as $\lceil \log_2 \max(\ell', c_0)\rceil = k$, the induction hypothesis implies that Line 14 performs at most $(c-1)(\lceil \log_2 \min(\ell, c)\rceil + 1) + 2(2^k - 1)$ additions, and at most $(c-1)\lceil \log_2 \min(\ell, c/2)\rceil/2 + 2(2^k - 1)$ multiplications. Finally, Lines 15 and 16 perform $\max(\ell'' - c, 0) \leq \ell'' = \ell - 2^k$ additions and multiplications. Summing these bounds, it follows that Algorithm 3 performs at most $(c-1)(\lceil \log_2 \min(\ell, c)\rceil + 1) + 2(\ell - 1)$ additions, and at most $(c-1)\lceil \log_2 \min(\ell, c/2)\rceil/2 + 2(\ell-1)$ multiplications.

Suppose now that $c_1 > 0$. Then Lines 3 to 9 of the algorithm perform $2\ell''$ additions and ℓ'' multiplications. As $\lceil \log_2 \min(\ell', c_1)\rceil \leq k$ and $\lceil \log_2 \min(\ell', c_1/2)\rceil \leq \lceil \log_2 \min(\ell', c_1)\rceil \leq \lceil \log_2 \ell'\rceil$, the induction hypothesis implies that Line 11 performs at most

$$(c_1 - 1)\left(\lceil \log_2 \ell'\rceil + 1\right) + 2\left(\ell' - 1\right) + \lceil \log_2 \ell'\rceil$$

additions, and at most $(c_1 - 1)\lceil \log_2 \ell'\rceil/2 + 2(\ell' - 1)$ multiplications. Lines 12 and 13 perform $\max(\ell'' - c_1, 0)$ additions. As $\ell' \leq 2^k$ and $c_0 = 2^k$, Lemma 4.4 implies that Line 14 performs at most $(c_0 - 1)(\lceil \log_2 \ell'\rceil + 1)$ additions, and at most $c_0\lceil \log_2 \ell'\rceil/2$ multiplications. Finally, Lines 15 and 16 perform no operations in $\mathbb{F}$, since $\ell'' \leq 2^k = c_0$. As $\ell' + \ell'' = \ell$ and $c_0 + c_1 = c$, it follows by summing these bounds that the algorithm performs at most

$$(c-1)\left(\lceil \log_2 \ell'\rceil + 1\right) + 2\left(\ell - 1\right) + \max\left(\ell'' - c_1, 0\right)$$

additions, and at most $(c-1)\lceil \log_2 \ell' \rceil/2 + 2(\ell-1)$ multiplications. If $\ell \leq 2^k$, then $\ell'' = 0$ and $\lceil \log_2 \ell' \rceil = \lceil \log_2 \min(\ell, c) \rceil = \lceil \log_2 \min(\ell, c/2) \rceil$. If $\ell > 2^k$, then $\ell'' = \ell - 2^k \leq 2^k \leq c-1$ and $\lceil \log_2 \ell' \rceil = \lceil \log_2 \min(\ell, c) \rceil - 1 \leq \lceil \log_2 \min(\ell, c/2) \rceil$. In either case, we find that Algorithm 3 performs at most $(c-1)(\lceil \log_2 \min(\ell, c) \rceil + 1) + 2(\ell-1)$ additions, and at most $(c-1)\lceil \log_2 \min(\ell, c/2) \rceil/2 + 2(\ell-1)$ multiplications. □

4.3. Truncated interpolation on the LCH basis

When Algorithm 3 is called with $c \leq \ell$, the input and output requirements on the vector $(a_i)_{0 \leq i < \max(\ell, c)} = (a_i)_{0 \leq i < \ell}$ define an invertible linear transformation from $\mathbb{F}^\ell$ onto $\mathbb{F}^\ell$. Inverting this transformation corresponds to solving an interpolation problem that involves taking a combination of a polynomial's evaluations and higher-degree coefficients on the LCH basis, and recovering its remaining coefficients. In this section, we deduce from Algorithm 3 an algorithm for solving the interpolation problem.

When Algorithm 3 is called with $c \leq \ell$, the variables defined in Lines 2 and 10 of the algorithm satisfy $k = \lceil \log_2 \ell \rceil - 1$, $\ell' = 2^k$, $\ell'' = \ell - 2^k$, $c_0 = \min(c, 2^k) \leq \ell'$, $c_1 = \max(c - 2^k, 0) \leq \ell'' \leq \ell'$, $t' = 2^k$ and $t'' = \ell''$. Consequently, Lines 11 and 14 recursively call the algorithm with parameters c and ℓ that once again satisfy $c \leq \ell$. Thus, the same is true of the parameters for all subsequent recursive calls. Moreover, as $\ell'' \geq c_1 = \min(\ell', c_1)$, the for-loop of Lines 8 and 9 has no effect on the vector $(a_i)_{0 \leq i < \ell}$ for the initial call to the algorithm, and, thus, all recursive calls. As $\ell'' \leq 2^k$ and $\mathbb{F}$ has characteristic equal to two, each remaining for-loop in the algorithm performs a transformation on the vector $(a_i)_{0 \leq i < \ell}$ that is an involution. Thus, performing these transformations in reverse order inverts the overall transformation performed by the algorithm, yielding an interpolation algorithm with identical complexity and space requirements. Pseudocode for this interpolation algorithm is presented in Algorithm 4, where it is once again assumed that $X_{2^i}(\beta_j)$ for $0 \leq i < j < \lceil \log_2 \ell \rceil$ have been precomputed for the input value of ℓ.

Algorithm 4 LCHInterp$(\ell, c, (X_{2^i}(\lambda))_{0 \leq i < \lceil \log_2 \ell \rceil}, (a_i)_{0 \leq i < \ell})$

Input: integers $\ell, c \in \{1, \ldots, 2^n\}$ such that $c \leq \ell$; the vector $(X_{2^i}(\lambda))_{0 \leq i < \lceil \log_2 \ell \rceil}$ for some $\lambda \in \mathbb{F}$; and the vector $(a_i)_{0 \leq i < \ell}$ such that for elements $h_0, \ldots, h_{\ell-1} \in \mathbb{F}$, $a_i = f(\omega_i + \lambda)$ for $i \in \{0, \ldots, c-1\}$, where $f = \sum_{i=0}^{\ell-1} h_i X_i$, and, if $c < \ell$, $a_i = h_i$ for $i \in \{c, \ldots, \ell-1\}$

Output: $a_i = h_i$ for $i \in \{0, \ldots, \ell-1\}$.

1: **if** $\ell = 1$ **then return**
2: $k \leftarrow \lceil \log_2 \ell \rceil - 1$, $\ell'' \leftarrow \ell - 2^k$, $c_0 \leftarrow \min(c, 2^k)$, $c_1 \leftarrow c - c_0$
3: **for** $i = c_0, \ldots, \ell'' - 1$ **do**
4: $\quad a_i \leftarrow a_i + X_{2^k}(\lambda) a_{2^k + i}$
5: LCHInterp$(2^k, c_0, (X_{2^i}(\lambda))_{0 \leq i < k}, (a_i)_{0 \leq i < 2^k})$
6: **if** $c_1 > 0$ **then**
7: $\quad$ **for** $i = c_1, \ldots, \ell'' - 1$ **do**
8: $\quad\quad a_{2^k + i} \leftarrow a_i + a_{2^k + i}$
9: $\quad$ LCHInterp$(2^k, c_1, (X_{2^i}(\lambda) + X_{2^i}(\beta_k))_{0 \leq i < k}, (a_i)_{2^k \leq i < \ell} \,\|\, (a_i)_{\ell'' \leq i < 2^k})$
10: $\quad$ **for** $i = 0, \ldots, \ell'' - 1$ **do**
11: $\quad\quad a_{2^k + i} \leftarrow a_i + a_{2^k + i}$
12: **for** $i = 0, \ldots, \ell'' - 1$ **do**
13: $\quad a_i \leftarrow a_i + X_{2^k}(\lambda) a_{2^k + i}$

Similar to Line 11 of Algorithm 3, Line 9 of Algorithm 4 can be realised by first cyclically shifting the subvector $(a_i)_{\ell''\leq i<\ell}$ left by $2^k-\ell''$ positions, then passing it to the recursive call, and finally cyclically shifting it right by $2^k-\ell''$ positions. As $2^k-\ell''=2^{\lceil\log_2\ell\rceil}-\ell$, the cyclic shifts are only required if ℓ is not a power of two, and, consequently, are not required by any recursive calls made by the algorithm. Using techniques already discussed for Algorithm 3, Algorithm 4 can be implemented so that only $O(\log^2\ell)$ field elements are required to be stored in auxiliary space. Similarly, all precomputations can be performed with $O(\log^2\ell)$ field operations. Algorithms 3 and 4 perform the same number of additions and multiplications in $\mathbb{F}$ when given identical inputs ℓ and c. Therefore, by substituting the inequality $c\leq\ell$ into the bounds of Theorem 4.3, we deduce the following bounds on the number of field operations performed by Algorithm 4.

Corollary 4.5. *Algorithm 4 performs at most* $(c-1)(\lceil\log_2 c\rceil+1)+2(\ell-1)$ *additions in* $\mathbb{F}$*, and at most* $(c-1)(\lceil\log_2 c\rceil-1)/2+2(\ell-1)$ *multiplications in* $\mathbb{F}$.

Remark 4.6. The ability to initially take $c<\ell$ in Algorithm 4 makes the algorithm suitable for use in the fast Hermite interpolation algorithm proposed by the author in (Coxon, 2020). For this application, one is given the higher order coefficients of a polynomial on the monomial basis, rather than on the LCH basis. However, Algorithm 8 of Section 5 can be used to compute the coefficients required by Algorithm 4. Similarly, Algorithm 3 can be combined with Algorithm 8 for use in the fast Hermite evaluation algorithm proposed in (Coxon, 2020).

5. Conversion between the monomial and LCH bases

We consider the problem of converting between the monomial and LCH bases under the assumption that there exists a tower of subfields

$$\mathbb{F}_2=\mathbb{F}_{2^{d_0}}\subset\mathbb{F}_{2^{d_1}}\subset\cdots\subset\mathbb{F}_{2^{d_m}}\subseteq\mathbb{F} \tag{5.1}$$

such that $d_{m-1}<n\leq d_m$, and $\beta_i/\beta_{d_t\lfloor i/d_t\rfloor}\in\mathbb{F}_{2^{d_t}}$ for $i\in\{0,\dots,n-1\}$ and $t\in\{0,\dots,m-1\}$. We can assume that $\mathbb{F}$ is not equal to $\mathbb{F}_2$, since otherwise $\beta=(1)$ and its associated LCH basis coincides with the monomial basis. Then the existence of such a tower is established by taking $m=1$ and $d_m=[\mathbb{F}:\mathbb{F}_2]$, so we have not imposed any restrictions on the vector β. However, our algorithms enjoy a reduction in their complexities when m is greater than one. To the tower we associate a family of bases of $\mathbb{F}[x]_{2^n}$ that includes the LCH basis and the twisted monomial basis $\{1,x/\beta_0,\dots,(x/\beta_0)^{2^n-1}\}$. We then show how to efficiently convert between its members, allowing for rapid conversion between the LCH and twisted monomial bases by traversing the family. Conversion between the LCH and monomial bases, in either direction, then requires only linearly many additional multiplications to be performed. We begin by introducing the family of bases and some supporting notation.

For $t\in\{0,\dots,m\}$, let q_t denote the order of the subfield $\mathbb{F}_{2^{d_t}}$. For $t\in\{0,\dots,m-1\}$ and $k\in\{0,\dots,\lceil n/d_t\rceil-1\}$, define $e_{t,k}=\lceil d_t(k+1)/d_{t+1}\rceil$ so that $d_{t+1}e_{t,k}$ is the least multiple of d_{t+1} that is greater than or equal to $d_t(k+1)$. For $t\in\{0,\dots,m-1\}$, $k\in\{0,\dots,\lceil n/d_t\rceil-1\}$ and $i\in\{0,\dots,2^n-1\}$, define

$$Y_i^{(t,k)}=\left(\prod_{s=0}^{k}X_{q_t^s}^{i_s}\right)\left(\prod_{s'=e_{t,k}}^{\lceil n/d_{t+1}\rceil-1}X_{q_{t+1}^{s'}}^{i'_{s'}}\right),$$

where

$$i_s=\sum_{r=0}^{d_t-1}2^r[i]_{d_ts+r},\quad i_k=\sum_{r=0}^{d_{t+1}e_{t,k}-d_tk-1}2^r[i]_{d_tk+r}\quad\text{and}\quad i'_{s'}=\sum_{r=0}^{d_{t+1}-1}2^r[i]_{d_{t+1}s'+r}$$

for $s \in \{0, \ldots, k-1\}$ and $s' \in \{e_{t,k}, \ldots, \lceil n/d_{t+1} \rceil - 1\}$. Then each polynomial $Y_i^{(t,k)}$ has degree equal to i. It follows that $\{Y_0^{(t,k)}, \ldots, Y_{\ell-1}^{(t,k)}\}$ is a basis of $\mathbb{F}[x]_\ell$ for $\ell \in \{1, \ldots, 2^n\}$, $t \in \{0, \ldots, m-1\}$ and $k \in \{0, \ldots, \lceil n/d_t \rceil - 1\}$. In Section 5.1, we show that the LCH basis is obtained by taking $t = 0$ and $k = \lceil n/d_0 \rceil - 1$, while the twisted monomial basis corresponds to $t = m-1$ and $k = 0$.

For $t \in \{0, \ldots, m-1\}$ and $k \in \{0, \ldots, \lceil n/d_t \rceil - 2\}$, define $\delta_{t,k} = X_{q_t^k}(\beta_{d_t(k+1)})^{q_t} - X_{q_t^k}(\beta_{d_t(k+1)})$. Then $\delta_{0,0}, \ldots, \delta_{0,n-2}$ are the denominators that appear in the recurrence relation from property (3) of Lemma 2.1. In Section 5.1, we show that our assumption on the quotients $\beta_i/\beta_{d_t\lfloor i/d_t \rfloor}$ leads to higher order recurrence relations between the polynomials $X_{2^0}, \ldots, X_{2^{n-1}}$ of the LCH basis in which the $\delta_{t,k}$ once again appear as denominators. For $t \in \{0, \ldots, m-1\}$, $k \in \{0, \ldots, \lceil n/d_t \rceil - 1\}$ and $\ell \in \{1, \ldots, 2^n\}$, define

$$I_{t,k,\ell} = \left\{0, \ldots, \min\left(\left\lceil \ell/q_t^k \right\rceil, q_{t+1}^{e_{t,k}}/q_t^k\right) - 1\right\},$$

and

$$J_{t,k,\ell} = \left\{j \in \{0, \ldots, \ell-1\} \mid [j]_{d_t k} = \cdots = [j]_{d_{t+1}e_{t,k}-1} = 0\right\}.$$

For $t \in \{0, \ldots, m-1\}$ and $k \in \{0, \ldots, \lceil n/d_t \rceil - 1\}$, the sets $\{iq_t^k + j \mid i \in I_{t,k,\ell-j}\}$ for $j \in J_{t,k,\ell}$ then form a partition of $\{0, \ldots, \ell-1\}$.

Having introduced the family of bases and the requisite notation, we are now ready to state the main technical lemma upon which we base our algorithms for converting between the LCH and monomial bases.

Lemma 5.1. *Let $f \in \mathbb{F}[x]_\ell$ for some $\ell \in \{1, \ldots, 2^n\}$. Then, for $t \in \{0, \ldots, m-1\}$ and $k \in \{0, \ldots, \lceil n/d_t \rceil - 1\}$, there exist unique elements $f_0^{(t,k)}, \ldots, f_{\ell-1}^{(t,k)} \in \mathbb{F}$ such that*

$$f = \sum_{i=0}^{\ell-1} f_i^{(t,k)} Y_i^{(t,k)}. \tag{5.2}$$

Moreover, the following properties hold for $t \in \{0, \ldots, m-1\}$ and $k \in \{0, \ldots, \lceil n/d_t \rceil - 1\}$:

1. *if $t = m-1$ and $k = 0$, then $f = \sum_{i=0}^{\ell-1} f_i^{(t,k)} (x/\beta_0)^i$,*
2. *if $k < \lceil n/d_t \rceil - 1$ and d_{t+1}/d_t divides $k+1$, then $f_i^{(t,k)} = f_i^{(t,k+1)}$ for $i \in \{0, \ldots, \ell-1\}$,*
3. *if $k < \lceil n/d_t \rceil - 1$ and d_{t+1}/d_t does not divide $k+1$, then*

$$\sum_{i=0}^{|I_{t,k,\ell-j}|-1} f_{iq_t^k+j}^{(t,k)} x^i = \sum_{i=0}^{|I_{t,k,\ell-j}|-1} f_{iq_t^k+j}^{(t,k+1)} x^{i-q_t\lfloor i/q_t \rfloor} \left(\frac{x^{q_t} - x}{\delta_{t,k}}\right)^{\lfloor i/q_t \rfloor}$$

   for $j \in J_{t,k,\ell}$, where $\delta_{t,k} = \mathrm{Tr}_{\mathbb{F}_{q_{t+1}}/\mathbb{F}_{q_t}}(\beta_{d_t(k+1)}/\beta_{d_t k})$ if $d_{t+1}/d_t = 2$,
4. *if $t > 0$ and $k \geq \lceil (\log_2 \ell)/d_t \rceil - 1$, then $f_i^{(t,k)} = f_i^{(t-1,0)}$ for $i \in \{0, \ldots, \ell-1\}$,*
5. *if $t = 0$ and $k \geq \lceil (\log_2 \ell)/d_t \rceil - 1$, then $f = \sum_{i=0}^{\ell-1} f_i^{(t,k)} X_i$.*

Lemma 5.1 is proved in Section 5.1. Properties (1) and (5) of the lemma identify the twisted monomial and LCH bases amongst the family of bases of $\mathbb{F}[x]_\ell$, while properties (2), (3) and (4) provide a means of traversing its members. Of the later three properties, only property (3) requires computation, which can be performed efficiently by applying the generalised Taylor expansion algorithm of Gao and Mateer (2010, Section II) or its inverse algorithm, depending on the direction of conversion. These algorithms are recalled in Section 5.3.

The number of times property (3) must be applied is invariant under the choice of tower:

$$\sum_{t=0}^{m-1} \sum_{\substack{k=0 \\ d_{t+1}/d_t \nmid k+1}}^{\lceil (\log_2 \ell)/d_t \rceil - 2} 1 = \sum_{t=0}^{m-1} \left(\left\lceil \frac{\log_2 \ell}{d_t} \right\rceil - \left\lceil \frac{\log_2 \ell}{d_{t+1}} \right\rceil \right) = \lceil \log_2 \ell \rceil - 1 \quad \text{for } \ell \in \{2, \ldots, q_m\}. \tag{5.3}$$

However, by using the algorithms of Gao and Mateer, the number of additions and multiplications required by a Taylor expansion step decreases as d_t grows. Thus, each subfield of degree less than $\log_2 \ell$ in the tower (5.1) effects the performance of our algorithms by reducing the number of additions and multiplications they perform. No multiplications are performed during a Taylor expansion step if $\delta_{t,k} = 1$, which in the case that $\mathbb{F}_{q_{t+1}}/\mathbb{F}_{q_t}$ is a quadratic extension occurs if and only if $\mathrm{Tr}_{\mathbb{F}_{q_{t+1}}/\mathbb{F}_{q_t}}(\beta_{d(k+1)}/\beta_{dk}) = 1$. Multiplications are also saved if $\beta_0 = 1$, since the twisted monomial basis coincides with the monomial basis. We cannot guarantee that any of these desirable properties are satisfied by an arbitrary choice of β. Furthermore, some of the properties are precluded when the field does not contain subfields of appropriate degree. However, the following proposition shows that when we have freedom to choose β, and subfields of appropriate degree do exist, then a standard basis construction can be used to obtain a vector with the desired properties.

Proposition 5.2. *Suppose there exists a tower of subfields $\mathbb{F}_2 = \mathbb{F}_{2^{d_0}} \subset \mathbb{F}_{2^{d_1}} \subset \cdots \subset \mathbb{F}_{2^{d_m}} \subseteq \mathbb{F}$. Let $\{\alpha_{t,0}, \ldots, \alpha_{t,d_{t+1}/d_t - 1}\}$ be a basis of $\mathbb{F}_{2^{d_{t+1}}}/\mathbb{F}_{2^{d_t}}$ for $t \in \{0, \ldots, m-1\}$, and*

$$\beta_i = \prod_{t=0}^{m-1} \alpha_{t,i_t} \quad \text{such that} \quad \sum_{t=0}^{m-1} i_t d_t = i$$

for $i \in \{0, \ldots, d_m - 1\}$. Then the following properties hold:

1. *$\beta_0, \ldots, \beta_{d_m - 1}$ are linearly independent over $\mathbb{F}_2$,*
2. *$\beta_i/\beta_{d_t \lfloor i/d_t \rfloor} \in \mathbb{F}_{2^{d_t}}$ for $i \in \{0, \ldots, d_m - 1\}$ and $t \in \{0, \ldots, m-1\}$,*
3. *if $\alpha_{0,0} = \cdots = \alpha_{m-1,0} = 1$, then $\beta_0 = 1$,*
4. *if $d_{t+1}/d_t = 2$ for some $t \in \{0, \ldots, m-1\}$, then $\beta_{d_t(k+1)}/\beta_{d_t k} = \alpha_{t,1}/\alpha_{t,0}$ for even $k \in \{0, \ldots, d_m/d_t - 2\}$.*

Proof. Let $i \in \{0, \ldots, d_m - 1\}$ and write $i = \sum_{t=0}^{m-1} i_t d_t$ with $i_t \in \{0, \ldots, d_{t+1}/d_t - 1\}$ for $t \in \{0, \ldots, m-1\}$. Then

$$\frac{\beta_i}{\beta_{d_t \lfloor i/d_t \rfloor}} = \frac{\alpha_{0,i_1} \cdots \alpha_{t-1,i_{t-1}} \alpha_{t,i_t} \cdots \alpha_{m-1,i_{m-1}}}{\alpha_{0,0} \cdots \alpha_{t-1,0} \alpha_{t,i_t} \cdots \alpha_{m-1,i_{m-1}}} = \frac{\alpha_{0,i_0} \cdots \alpha_{t-1,i_{t-1}}}{\alpha_{0,0} \cdots \alpha_{t-1,0}} \in \mathbb{F}_{2^{d_t}}$$

for $t \in \{0, \ldots, m-1\}$. Thus, property (2) holds. Similarly, we have $\beta_{d_t i + j}/\beta_0 = (\alpha_{t,i}/\alpha_{t,0})(\beta_j/\beta_0)$ and $\beta_j/\beta_0 = \beta_j/\beta_{d_t \lfloor j/d_t \rfloor} \in \mathbb{F}_{2^{d_t}}$ for $i \in \{0, \ldots, d_{t+1}/d_t - 1\}$, $j \in \{0, \ldots, d_t - 1\}$ and $t \in \{0, \ldots, m-1\}$. As $\{\alpha_{t,0}/\alpha_{t,0}, \ldots, \alpha_{t,d_{t+1}/d_t - 1}/\alpha_{t,0}\}$ is a basis of $\mathbb{F}_{2^{d_{t+1}}}/\mathbb{F}_{2^{d_t}}$ for $t \in \{0, \ldots, m-1\}$, it follows that if $\{\beta_0/\beta_0, \ldots, \beta_{d_t - 1}/\beta_0\}$ is a basis of $\mathbb{F}_{2^{d_t}}/\mathbb{F}_2$ for some $t \in \{0, \ldots, m-1\}$, then $\{\beta_0/\beta_0, \ldots, \beta_{d_{t+1} - 1}/\beta_0\}$ is a basis of $\mathbb{F}_{2^{d_{t+1}}}/\mathbb{F}_2$. For $t = 0$, it is clear that $\{\beta_0/\beta_0, \ldots, \beta_{d_t - 1}/\beta_0\} = \{1\}$ is a basis of $\mathbb{F}_{2^{d_t}}/\mathbb{F}_2 = \mathbb{F}_2/\mathbb{F}_2$. Therefore, $\{\beta_0/\beta_0, \ldots, \beta_{d_m - 1}/\beta_0\}$ is a basis of $\mathbb{F}_{2^{d_m}}/\mathbb{F}_2$, and property (1) holds.

By definition, β_0 is the product of $\alpha_{0,0}, \ldots, \alpha_{m-1,0}$, from which property (3) follows immediately. Property (4) holds since if $d_{t+1}/d_t = 2$ for some $t \in \{0, \ldots, m-1\}$, then $\beta_{d_t(k+1)}/\beta_{d_t k} = \beta_{d_{t+1}(k/2) + d_t}/\beta_{d_{t+1}(k/2)} = \alpha_{t,1}/\alpha_{t,0}$ for even $k \in \{0, \ldots, d_m/d_t - 2\}$. □

If the tower in Proposition 5.2 contains a quadratic extension $\mathbb{F}_{2^{d_{t+1}}}/\mathbb{F}_{2^{d_t}}$, then taking $\alpha_{t,0} = 1$ and $\alpha_{t,1} \in \mathbb{F}_{2^{d_{t+1}}}$ such that $\mathrm{Tr}_{\mathbb{F}_{2^{d_{t+1}}}/\mathbb{F}_{2^{d_t}}}(\alpha_{t,1}) = 1$ yields a basis $\{\alpha_{t,0}, \alpha_{t,1}\}$ of the extension with the property that $\alpha_{t,1}/\alpha_{t,0}$ has trace equal to one. Indeed, linear independence of the two elements over $\mathbb{F}_{2^{d_t}}$ follows from the observation that $\mathrm{Tr}_{\mathbb{F}_{2^{d_{t+1}}}/\mathbb{F}_{2^{d_t}}}(1) = 0$.

5.1. Proof of Lemma 5.1

We begin by establishing properties (1), (2), (4) and (5) of Lemma 5.1.

Lemma 5.3. *The following properties hold for $t \in \{0, \dots, m-1\}$ and $k \in \{0, \dots, \lceil n/d_t \rceil - 1\}$:*

1. *if $t = m-1$ and $k = 0$, then $Y_i^{(t,k)} = (x/\beta_0)^i$ for $i \in \{0, \dots, 2^n - 1\}$,*
2. *if $k < \lceil n/d_t \rceil - 1$ and d_{t+1}/d_t divides $k+1$, then $Y_i^{(t,k)} = Y_i^{(t,k+1)}$ for $i \in \{0, \dots, 2^n - 1\}$,*
3. *if $t > 0$, then $Y_i^{(t,k)} = Y_i^{(t-1,0)}$ for $i \in \{0, \dots, \min(q_t^{k+1}, 2^n) - 1\}$,*
4. *if $t = 0$, then $Y_i^{(t,k)} = X_i$ for $i \in \{0, \dots, 2^{k+1} - 1\}$.*

Proof. We have $\lceil n/d_m \rceil - 1 = 0$, $e_{m-1,0} = 1$ and

$$\sum_{r=0}^{d_m e_{m-1,0} - d_{m-1} \times 0 - 1} 2^r [i]_{d_{m-1} \times 0 + r} = \sum_{r=0}^{d_m - 1} 2^r [i]_r = i \quad \text{for } i \in \{0, \dots, 2^n - 1\}.$$

Thus, $Y_i^{(m-1,0)} = X_1^i$ for $i \in \{0, \dots, 2^n - 1\}$. As $X_1 = x/\beta_0$ by property (3) of Lemma 2.1, it follows that property (1) holds.

Let $t \in \{0, \dots, m-1\}$ and $k \in \{0, \dots, \lceil n/d_t \rceil - 2\}$ such that d_{t+1}/d_t divides $k+1$. Then $e_{t,k} = d_t(k+1)/d_{t+1}$ and $e_{t,k+1} = e_{t,k} + 1$. Thus, for $i \in \{0, \dots, 2^n - 1\}$,

$$\sum_{r=0}^{d_{t+1} e_{t,k} - d_t k - 1} 2^r [i]_{d_t k + r} = \sum_{r=0}^{d_t - 1} 2^r [i]_{d_t k + r} \quad \text{and} \quad \sum_{r=0}^{d_{t+1} - 1} 2^r [i]_{d_{t+1} e_{t,k} + r} = \sum_{r=0}^{d_{t+1} e_{t,k+1} - d_t(k+1) - 1} 2^r [i]_{d_t(k+1) + r}.$$

It follows that property (2) holds.

Suppose that $t \in \{1, \dots, m-1\}$ and $k \in \{0, \dots, \lceil n/d_t \rceil - 1\}$. Then $d_{t+1} e_{t,k} \geq d_t(k+1)$ and $e_{t-1,0} = 1$. For $i \in \{0, \dots, q_t^{k+1} - 1\}$, it follows that

$$\sum_{r=0}^{d_t - 1} 2^r [i]_{d_t s + r} = 0 \quad \text{for } s \in \{k+1, \dots, \lceil n/d_t \rceil - 1\}$$

and

$$\sum_{r=0}^{d_{t+1} - 1} 2^r [i]_{d_{t+1} s' + r} = 0 \quad \text{for } s' \in \{e_{t,k}, \dots, \lceil n/d_t \rceil - 1\}.$$

Thus, if $i \in \{0, \dots, \min(q_t^{k+1}, 2^n) - 1\}$ and $i_s = \sum_{r=0}^{d_t - 1} 2^r [i]_{d_t s + r}$ for $s \in \{0, \dots, \lceil n/d_t \rceil - 1\}$, then

$$i_0 = \sum_{r=0}^{d_t - 1} 2^r [i]_{d_t \times 0 + r} = \sum_{r=0}^{d_t e_{t-1,0} - d_{t-1} \times 0 - 1} 2^r [i]_{d_{t-1} \times 0 + r}$$

and

$$Y_i^{(t,k)} = \prod_{s=0}^{k} X_{q_t^s}^{i_s} = \left(\prod_{s=0}^{0} X_{q_{t-1}^s}^{i_s} \right) \left(\prod_{s'=e_{t-1,0}}^{\lceil n/d_t \rceil - 1} X_{q_t^{s'}}^{i_{s'}} \right) = Y_i^{(t-1,0)}.$$

Therefore, property (3) holds.

As $d_0 = 1$, we have $d_1 e_{0,k} = d_1\lceil (k+1)/d_1 \rceil \geq k+1$ for $k \in \{0, \ldots, \lceil n/d_1 \rceil - 1\}$. Consequently, property (1) of Lemma 2.1 implies that

$$Y_i^{(0,k)} = \left(\prod_{s=0}^{k} X_{2^s}^{[i]_s} \right) \left(\prod_{s'=e_{0,k}}^{\lceil n/d_1 \rceil - 1} X_{q_1^{s'}}^{0} \right) = \prod_{s=0}^{k} X_{2^s [i]_s} = X_i$$

for $k \in \{0, \ldots, \lceil n/d_1 \rceil - 1\}$ and $i \in \{0, \ldots, 2^{k+1} - 1\}$. Hence, property (4) holds. □

Recall that a polynomial in $\mathbb{F}[x]$ is $\mathbb{F}_q$-linearised if it can be written in the form $\sum_{i=0}^{k} f_i x^{q^i}$ with $f_0, \ldots, f_k \in \mathbb{F}$. The following lemma generalises properties (3) and (5) of Lemma 2.1, and shows that our assumption on the quotients $\beta_i/\beta_{d_t \lfloor i/d_t \rfloor}$ leads to higher order and, importantly, sparse recurrence relations between the polynomials $X_{2^0}, \ldots, X_{2^{n-1}}$ of the LCH basis.

Lemma 5.4. *Let $\mathbb{F}_{2^d} \subseteq \mathbb{F}$ such that $\beta_i/\beta_{d\lfloor i/d \rfloor} \in \mathbb{F}_{2^d}$ for $i \in \{0, \ldots, n-1\}$. Then $X_{2^{dk}}$ is $\mathbb{F}_{2^d}$-linearised $k \in \{0, \ldots, \lceil n/d \rceil - 1\}$, and*

$$X_{2^{d(k+1)}} = \frac{X_{2^{dk}}(x)^{2^d} - X_{2^{dk}}(x)}{X_{2^{dk}}(\beta_{d(k+1)})^{2^d} - X_{2^{dk}}(\beta_{d(k+1)})} \tag{5.4}$$

for $k \in \{0, \ldots, \lceil n/d \rceil - 2\}$. Moreover, if $\mathbb{F}_{2^{2d}} \subseteq \mathbb{F}$ and $\beta_i/\beta_{2d\lfloor i/(2d) \rfloor} \in \mathbb{F}_{2^{2d}}$ for $i \in \{0, \ldots, n-1\}$, then

$$X_{2^{dk}}(\beta_{d(k+1)})^{2^d} - X_{2^{dk}}(\beta_{d(k+1)}) = \mathrm{Tr}_{\mathbb{F}_{2^{2d}}/\mathbb{F}_{2^d}}(\beta_{d(k+1)}/\beta_{dk}) \tag{5.5}$$

for even $k \in \{0, \ldots, \lceil n/d \rceil - 2\}$.

Proof. Let $\mathbb{F}_{2^d} \subseteq \mathbb{F}$ such that $\beta_i/\beta_{d\lfloor i/d \rfloor} \in \mathbb{F}_{2^d}$ for $i \in \{0, \ldots, n-1\}$. We show that if $X_{2^{dk}}$ is $\mathbb{F}_{2^d}$-linearised for some $k \in \{0, \ldots, \lceil n/d \rceil - 2\}$, then (5.4) holds and $X_{2^{d(k+1)}}$ is $\mathbb{F}_{2^d}$-linearised. As property (3) of Lemma 2.1 implies that $X_{2^{d\cdot 0}} = x/\beta_0$ is $\mathbb{F}_{2^d}$-linearised, the first assertion of the lemma then follows by induction on k.

Suppose that $X_{2^{dk}}$ is $\mathbb{F}_{2^d}$-linearised for some $k \in \{0, \ldots, \lceil n/d \rceil - 2\}$. Then

$$\frac{\omega_{2^{dk} i}}{\beta_{dk}} = \sum_{j=0}^{d-1} [i]_j \frac{\beta_{dk+j}}{\beta_{dk}} = \sum_{j=0}^{d-1} [i]_j \frac{\beta_{dk+j}}{\beta_{d\lfloor (dk+j)/d \rfloor}} \in \mathbb{F}_{2^d} \quad \text{for } i \in \{0, \ldots, 2^d - 1\}.$$

Thus, property (5) of Lemma 2.1 and the assumption that $X_{2^{dk}}$ is $\mathbb{F}_{2^d}$-linearised imply that

$$X_{2^{dk}}\left(\omega_{2^{dk} i + j}\right) = X_{2^{dk}}\left(\beta_{dk} \frac{\omega_{2^{dk} i}}{\beta_{dk}}\right) + X_{2^{dk}}(\omega_j) = X_{2^{dk}}(\beta_{dk}) \frac{\omega_{2^{dk} i}}{\beta_{dk}} + X_{2^{dk}}(\omega_j) = \frac{\omega_{2^{dk} i}}{\beta_{dk}} \in \mathbb{F}_{2^d}$$

for $i \in \{0, \ldots, 2^d - 1\}$ and $j \in \{0, \ldots, 2^{dk} - 1\}$. It follows that ω_i is a common root of the degree $2^{d(k+1)}$ polynomials $X_{2^{d(k+1)}}$ and $X_{2^{dk}}^{2^d} - X_{2^{dk}}$ for $i \in \{0, \ldots, 2^{d(k+1)} - 1\}$. Hence, they are equal up to a nonzero scalar multiple. As $X_{2^{d(k+1)}}(\beta_{d(k+1)}) = 1$, it follows that (5.4) holds. Equation (5.4) then implies that $X_{2^{d(k+1)}}$ is $\mathbb{F}_{2^d}$-linearised, since $X_{2^{dk}}$ is $\mathbb{F}_{2^d}$-linearised by assumption.

Suppose now that $\mathbb{F}_{2^{2d}} \subseteq \mathbb{F}$ and $\beta_i/\beta_{2d\lfloor i/(2d) \rfloor} \in \mathbb{F}_{2^{2d}}$ for $i \in \{0, \ldots, n-1\}$. Then, for even $k \in \{0, \ldots, \lceil n/d \rceil - 2\}$, $X_{2^{dk}} = X_{2^{2d(k/2)}}$ is $\mathbb{F}_{2^{2d}}$-linearised and

$$\frac{\beta_{d(k+1)}}{\beta_{dk}} = \frac{\beta_{d(k+1)}}{\beta_{2d(k/2)}} = \frac{\beta_{d(k+1)}}{\beta_{2d\lfloor d(k+1)/(2d) \rfloor}} \in \mathbb{F}_{2^{2d}}.$$

Hence,

$$X_{2^{dk}}(\beta_{d(k+1)}) = X_{2^{dk}}\left(\beta_{dk}\frac{\beta_{d(k+1)}}{\beta_{dk}}\right) = X_{2^{dk}}(\beta_{dk})\frac{\beta_{d(k+1)}}{\beta_{dk}} = \frac{\beta_{d(k+1)}}{\beta_{dk}}$$

for even $k \in \{0, \ldots, \lceil n/d \rceil - 2\}$, from which (5.5) follows. □

If $t \in \{0, \ldots, m-1\}$ and $k \in \{0, \ldots, \lceil n/d_t \rceil - 2\}$ such that d_{t+1}/d_t does not divide $k+1$, then $e_{t,k+1} = e_{t,k}$. It follows in this case that each quotient $Y_i^{(t,k+1)}/Y_i^{(t,k)}$ is of the form $X_{q_t^{k+1}}^a / X_{q_t^k}^b$ for some $a, b \in \mathbb{N}$. Using Lemma 5.4 to express these quotients as a functions of $X_{q_t^k}$ leads to property (3) of Lemma 5.1.

Lemma 5.5. *If $t \in \{0, \ldots, m-1\}$ and $k \in \{0, \ldots, \lceil n/d_t \rceil - 2\}$ such that d_{t+1}/d_t does not divide $k+1$, then*

$$Y_{iq_t^k+j}^{(t,k)} = X_{q_t^k}^i Y_j^{(t,k)} \quad \text{and} \quad Y_{iq_t^k+j}^{(t,k+1)} = \left(\frac{X_{q_t^k}^{q_t} - X_{q_t^k}}{\delta_{t,k}}\right)^{\lfloor i/d_t \rfloor} X_{q_t^k}^{i - q_t\lfloor i/d_t \rfloor} Y_j^{(t,k)}$$

for $i \in I_{t,k,2^n}$ and $j \in J_{t,k,2^n}$.

Proof. Suppose that $t \in \{0, \ldots, m-1\}$ and $k \in \{0, \ldots, \lceil n/d_t \rceil - 2\}$ such that d_{t+1}/d_t does not divide $k+1$. Then, for $i \in I_{t,k,2^n}$ and $j \in J_{t,k,2^n}$, we have

$$\sum_{r=0}^{d_t-1} 2^r [iq_t^k + j]_{d_t s + r} = \sum_{r=0}^{d_t-1} 2^r [j]_{d_t s + r} \quad \text{for } s \in \{0, \ldots, k-1\},$$

$$\sum_{r=0}^{d_{t+1}e_{t,k} - d_t k - 1} 2^r [iq_t^k + j]_{d_t k + r} = \sum_{r=0}^{d_{t+1}e_{t,k} - d_t k - 1} 2^r [i]_r = i + \sum_{r=0}^{d_{t+1}e_{t,k} - d_t k - 1} 2^r [j]_{d_t k + r}$$

and

$$\sum_{r=0}^{d_{t+1}-1} 2^r [iq_t^k + j]_{d_{t+1}s' + r} = \sum_{r=0}^{d_{t+1}-1} 2^r [j]_{d_{t+1}s' + r} \quad \text{for } s' \in \{e_{t,k}, \ldots, \lceil n/d_t \rceil - 1\}.$$

Consequently, $Y_{iq_t^k+j}^{(t,k)} = X_{q_t^k}^i Y_j^{(t,k)}$ for $i \in I_{t,k,2^n}$ and $j \in J_{t,k,2^n}$.

As d_{t+1}/d_t does not divide $k+1$, $e_{t,k+1} = e_{t,k}$ and $d_{t+1}e_{t,k+1} = d_{t+1}e_{t,k} > d_t(k+1)$. Thus, for $i \in I_{t,k,2^n}$ and $j \in J_{t,k,2^n}$, we have

$$\sum_{r=0}^{d_t-1} 2^r [iq_t^k + j]_{d_t k + r} = \sum_{r=0}^{d_t-1} 2^r [i]_r = \left(i - q_t\left\lfloor \frac{i}{q_t} \right\rfloor\right) + \sum_{r=0}^{d_t-1} 2^r [j]_{d_t k + r}$$

and

$$\sum_{r=0}^{d_{t+1}e_{t,k+1} - d_t(k+1) - 1} 2^r [iq_t^k + j]_{d_t(k+1) + r} = \sum_{r=0}^{d_{t+1}e_{t,k} - d_t(k+1) - 1} 2^r [i]_{d_t + r} = \left\lfloor \frac{i}{q_t} \right\rfloor + \sum_{r=0}^{d_{t+1}e_{t,k} - d_t(k+1) - 1} 2^r [j]_{d_t(k+1) + r}.$$

It follows that

$$Y_{iq_t^k+j}^{(t,k+1)} = X_{q_t^{k+1}}^{\lfloor i/q_t \rfloor} X_{q_t^k}^{i - q_t\lfloor i/q_t \rfloor} Y_j^{(t,k)} \quad \text{for } i \in I_{t,k,2^n} \text{ and } j \in J_{t,k,2^n}.$$

The proof is completed by making the substitution $X_{q_t^{k+1}} = (X_{q_t^k}^{q_t} - X_{q_t^k})/\delta_{t,k}$, which holds by Lemma 5.4 and the assumption that $\beta_i/\beta_{d_t\lfloor i/d_t \rfloor} \in \mathbb{F}_{q_t}$ for $i \in \{0, \ldots, n-1\}$. □

We are now ready to prove Lemma 5.1.

Proof of Lemma 5.1. Let $\ell \in \{1, \dots, 2^n\}$ and $f \in \mathbb{F}[x]_\ell$. Then, for $t \in \{0, \dots, m-1\}$ and $k \in \{0, \dots, \lceil n/d_t \rceil - 1\}$, existence and uniqueness of the elements $f_0^{(t,k)}, \dots, f_{\ell-1}^{(t,k)}$ follows from the observation that $Y_i^{(t,k)}$ has degree equal to i for $i \in \{0, \dots, \ell-1\}$. Properties (1), (2), (4) and (5) then follow immediately from Lemma 5.3. To prove property (3), suppose that $t \in \{0, \dots, m-1\}$ and $k \in \{0, \dots, \lceil n/d_t \rceil - 2\}$ such that d_{t+1}/d_t does not divide $k+1$. Then k is even if $d_{t+1}/d_t = 2$, in which case Lemma 5.4 implies that $\delta_{t,k} = \mathrm{Tr}_{\mathbb{F}_{q_{t+1}}/\mathbb{F}_{q_t}}(\beta_{d_t(k+1)}/\beta_{d_t k})$. The sets $\{iq_t^k + j \mid i \in I_{t,k,\ell-j}\} = \{iq_t^k + j \mid i \in \{0, \dots, |I_{t,k,\ell-j}| - 1\}\}$ for $j \in J_{t,k,\ell}$ form a partition of $\{0, \dots, \ell-1\}$. Thus, there exist elements $f'_0, \dots, f'_{\ell-1} \in \mathbb{F}$ such that

$$\sum_{i=0}^{|I_{t,k,\ell-j}|-1} f_{iq_t^k+j}^{(t,k)} x^j = \sum_{i=0}^{|I_{t,k,\ell-j}|-1} f'_{iq_t^k+j} x^{j-q_t\lfloor i/q_t \rfloor} \left(\frac{x^{q_t} - x}{\delta_{t,k}}\right)^{\lfloor i/q_t \rfloor} \quad \text{for } j \in J_{t,k,\ell}.$$

After substituting $X_{q_t^k}$ for x, then multiplying each equation by $Y_j^{(t,k)}$, Lemma 5.5 implies that

$$\sum_{i=0}^{|I_{t,k,\ell-j}|-1} f_{iq_t^k+j}^{(t,k)} Y_{iq_t^k+j}^{(t,k)} = \sum_{i=0}^{|I_{t,k,\ell-j}|-1} f'_{iq_t^k+j} Y_{iq_t^k+j}^{(t,k+1)} \quad \text{for } j \in J_{t,k,\ell}.$$

By summing these equations, it follows that

$$f = \sum_{j \in J_{t,k,\ell}} \sum_{i \in I_{t,k,\ell-j}} f_{iq_t^k+j}^{(t,k)} Y_{iq_t^k+j}^{(t,k)} = \sum_{i=0}^{\ell-1} f'_i Y_i^{(t,k+1)}.$$

Thus, the uniqueness of $f_0^{(t,k+1)}, \dots, f_{\ell-1}^{(t,k+1)}$ implies that $f'_i = f_i^{(t,k+1)}$ for $i \in \{0, \dots, \ell-1\}$. Hence, property (3) follows by the choice of $f'_0, \dots, f'_{\ell-1}$. □

5.2. Precomputations

The algorithm we propose for converting from the monomial basis to the LCH basis assumes that $\delta_{t,k}$ has been precomputed for $t \in \{0, \dots, m-1\}$ and $k \in \{0, \dots, \lceil (\log_2 \ell)/d_t \rceil - 2\}$ such that d_{t+1}/d_t does not divide $k+1$. The algorithm for conversion in the opposite direction assumes that the inverses of these elements and β_0 have been precomputed. Thus, (5.3) implies that each algorithm requires $O(\log \ell)$ field elements to be precomputed and stored. The precomputations for both algorithms can be performed by applying Algorithm 5 for each $d \in \{d_0, \dots, d_{m-1}\}$. The algorithm performs $O((\log^2 \ell)/d)$ operations in $\mathbb{F}$ if repeated squaring is used for exponentiation. As $d_t \geq 2^t$ for $t \in \{0, \dots, m-1\}$, it follows that all precomputations for the conversion algorithms can be performed with $O(\log^2 \ell)$ operations in $\mathbb{F}$.

As $X_{2^0} = x/\beta_0$, Lines 1 to 3 of Algorithm 5 set $x_j = X_{2^{d \cdot 0}}(\beta_{d(j+1)})$ for $j \in \{0, \dots, \lceil (\log_2 \ell)/d \rceil - 2\}$. Using induction and Lemma 5.4, it can then been shown that during iteration i of the outer for-loop of Lines 4 to 7, Line 5 sets $x_i = X_{2^{di}}(\beta_{d(i+1)})^{2^d} - X_{2^{di}}(\beta_{d(i+1)})$, and Lines 6 and 7 set $x_j = X_{2^{d(i+1)}}(\beta_{d(j+1)})$ for $j \in \{i+1, \dots, \lceil (\log_2 \ell)/d \rceil - 2\}$. Correctness of the algorithm then follows.

Remark 5.6. Algorithm 5 is readily modified to perform the precomputations required by the algorithms of Sections 3 and 4 by setting $d = 1$, taking $k = \lceil \log_2 \ell \rceil - 1$ in Line 1, and storing the intermediate values of $x_0, \dots, x_k$ computed in Lines 3 and 7. The entries of $(X_{2^i}(\lambda))_{0 \leq i < \lceil \log_2 \ell \rceil}$ for some $\lambda \in \mathbb{F}$ can be computed alongside the algorithm with $O(\log \ell)$ operations in $\mathbb{F}$ as the sequence $X_{2^0}(\lambda) = \lambda y$ and $X_{2^i}(\lambda) = (X_{2^{i-1}}(\lambda)^2 - X_{2^{i-1}}(\lambda))y_{i-1}$ for $i \in \{1, \dots, \lceil \log_2 \ell \rceil - 1\}$. Thus, it is worthwhile to store y and $y_0, \dots, y_{\lceil \log_2 \ell \rceil - 2}$ if the vector has to be computed for several different λ.

Algorithm 5 Precomputations(d, ℓ)

Input: $d \in \mathbb{N}$ such that $\mathbb{F}_{2^d} \subseteq \mathbb{F}$ and $\beta_i/\beta_{d\lfloor i/d \rfloor} \in \mathbb{F}_{2^d}$ for $i \in \{0, \dots, n-1\}$, and $\ell \in \{1, \dots, 2^n\}$.
Output: $X_{2^{di}}(\beta_{d(i+1)})^{2^d} - X_{2^{di}}(\beta_{d(i+1)})$ and its inverse for $i \in \{0, \dots, \lceil (\log_2 \ell)/d \rceil - 2\}$, and $1/\beta_0$.

1: $k \leftarrow \lceil (\log_2 \ell)/d \rceil - 2$, $y \leftarrow 1/\beta_0$
2: **for** $j = 0, \dots, k$ **do**
3: $\quad x_j \leftarrow \beta_{d(j+1)} y$
4: **for** $i = 0, \dots, k$ **do**
5: $\quad x_i \leftarrow x_i^{2^d} - x_i$, $y_i \leftarrow 1/x_i$
6: $\quad$ **for** $j = i+1, \dots, k$ **do**
7: $\quad\quad x_j \leftarrow (x_j^{2^d} - x_j) y_i$
8: **return** x_i and y_i for $i \in \{0, \dots, k\}$, and y

5.3. The generalised Taylor expansion algorithm of Gao and Mateer

The generalised Taylor expansion of a polynomial $f \in \mathbb{F}[x]$ at a nonconstant polynomial $p \in \mathbb{F}[x]$ is the series expansion $f = f_0 + f_1 p + f_2 p^2 + \cdots$ such that $f_i \in \mathbb{F}[x]_{\deg p}$ for $i \in \mathbb{N}$. Gao and Mateer (2010, Section II) propose a quasi-linear time algorithm for computing the coefficients of the Taylor expansion when $p = x^t - x$ for some $t \geq 2$. We present a nonrecursive version of their algorithm for t equal to a power of two, along with a matching algorithm for recovering polynomials from the coefficients of their Taylor expansion. Finally, we derive a bound on the complexity of both algorithms which is tighter than the one provided by Gao and Mateer.

Let $d, \ell \in \mathbb{N}$ be nonzero and $f \in \mathbb{F}[x]_\ell$. For $k \in \mathbb{N}$, let $f_{k,0}, f_{k,1}, \dots \in \mathbb{F}[x]_{2^{d+k}}$ be the coefficients of the Taylor expansion of f at $(x^{2^d} - x)^{2^k}$. Then $f_{k,i} = 0$ if $i \geq \lceil \ell/2^{d+k} \rceil$, and $f_{k,0} = f$ if $k \geq \lceil \log_2 \lceil \ell/2^d \rceil \rceil$. Moreover,

$$f = \sum_{i \in \mathbb{N}} \left(f_{k,2i} + (x^{2^d} - x)^{2^k} f_{k,2i+1} \right) \left(x^{2^d} - x \right)^{2^{k+1} i} \quad \text{for } k \in \mathbb{N}.$$

Thus, $f_{k+1,i} = f_{k,2i} + x^{2^k} f_{k,2i+1} + x^{2^{d+k}} f_{k,2i+1}$ for $k, i \in \mathbb{N}$. Given the coefficients of $f_{k,2i}$ and $f_{k,2i+1}$ on the monomial basis, it follows that the coefficients of $f_{k+1,i}$ on the monomial basis can be computed with 2^{d+k} additions. This computation is also readily inverted by performing the same number of additions. Consequently, given the Taylor coefficients $f_{0,0}, f_{0,1}, \dots, f_{0,\lceil \ell/2^d \rceil - 1}$ with respect to the monomial basis, we can efficiently compute $f = f_{\lceil \log_2 \lceil \ell/2^d \rceil \rceil, 0}$ with respect to the monomial basis by means of the recursive formula, and vice versa. Using this observation, we obtain Algorithms 6 and 7.

Lemma 5.7. *Algorithms 6 and 7 perform at most* $\lfloor \ell/2 \rfloor \lceil \log_2 \lceil \ell/2^d \rceil \rceil$ *additions in* $\mathbb{F}$.

Proof. For each $k \in \{0, \dots, \lceil \log_2 \lceil \ell/2^d \rceil \rceil - 1\}$, Lines 2–7 of either algorithm perform

$$2^{d+k} \ell_1 + \max\left(\ell_2 - 2^{d+k}, 0 \right) \leq 2^{d+k} \ell_1 + (\ell_2 - \lceil \ell_2/2 \rceil) = 2^{d+k} \ell_1 + \lfloor \ell_2/2 \rfloor = \lfloor \ell/2 \rfloor$$

additions in $\mathbb{F}$. □

5.4. Conversion algorithms

By combining Lemma 5.1 with the Taylor expansion algorithms of Section 5.3, we obtain Algorithms 8 and 9 for converting between the monomial and LCH bases. Each algorithm operates on a vector $(a_i)_{0 \leq i < \ell}$ which initially contains the coefficients of a polynomial on the input

Algorithm 6 TaylorExpansion$(d, \ell, (a_i)_{0\le i<\ell})$

Input: positive integers d and ℓ; and the vector $(a_i)_{0\le i<\ell}$ such that $a_i = f_i \in \mathbb{F}$ for $i \in \{0,\dots,\ell-1\}$.
Output: $a_i = h_i \in \mathbb{F}$ for $i \in \{0,\dots,\ell-1\}$ such that $\sum_{i=0}^{\ell-1} h_i x^{i-2^d\lfloor i/2^d\rfloor}(x^{2^d}-x)^{\lfloor i/2^d\rfloor} = \sum_{i=0}^{\ell-1} f_i x^i$.

1: **for** $k = \lceil \log_2 \lceil \ell/2^d \rceil \rceil - 1, \dots, 0$ **do**
2: $\quad \ell_1 \leftarrow \lfloor \ell/(2^{d+k+1}) \rfloor$, $\ell_2 \leftarrow \ell - 2^{d+k+1}\ell_1$
3: $\quad$ **for** $i = 0, \dots, \ell_1 - 1$ **do**
4: $\quad\quad$ **for** $j = 2^{d+k} - 1, \dots, 0$ **do**
5: $\quad\quad\quad a_{2^{d+k}(2i)+2^k+j} \leftarrow a_{2^{d+k}(2i)+2^k+j} + a_{2^{d+k}(2i+1)+j}$
6: $\quad$ **for** $j = \ell_2 - 2^{d+k} - 1, \dots, 0$ **do**
7: $\quad\quad a_{2^{d+k}(2\ell_1)+2^k+j} \leftarrow a_{2^{d+k}(2\ell_1)+2^k+j} + a_{2^{d+k}(2\ell_1+1)+j}$

Algorithm 7 InverseTaylorExpansion$(d, \ell, (a_i)_{0\le i<\ell})$

Input: positive integers d and ℓ; and the vector $(a_i)_{0\le i<\ell}$ such that $a_i = h_i \in \mathbb{F}$ for $i \in \{0,\dots,\ell-1\}$.
Output: $a_i = f_i$ for $i \in \{0,\dots,\ell-1\}$ such that $\sum_{i=0}^{\ell-1} f_i x^i = \sum_{i=0}^{\ell-1} h_i x^{i-2^d\lfloor i/2^d\rfloor}(x^{2^d}-x)^{\lfloor i/2^d\rfloor}$.

1: **for** $k = 0, \dots, \lceil \log_2 \lceil \ell/2^d \rceil \rceil - 1$ **do**
2: $\quad \ell_1 \leftarrow \lfloor \ell/(2^{d+k+1}) \rfloor$, $\ell_2 \leftarrow \ell - 2^{d+k+1}\ell_1$
3: $\quad$ **for** $i = 0, \dots, \ell_1 - 1$ **do**
4: $\quad\quad$ **for** $j = 0, \dots, 2^{d+k} - 1$ **do**
5: $\quad\quad\quad a_{2^{d+k}(2i)+2^k+j} \leftarrow a_{2^{d+k}(2i)+2^k+j} + a_{2^{d+k}(2i+1)+j}$
6: $\quad$ **for** $j = 0, \dots, \ell_2 - 2^{d+k} - 1$ **do**
7: $\quad\quad a_{2^{d+k}(2\ell_1)+2^k+j} \leftarrow a_{2^{d+k}(2\ell_1)+2^k+j} + a_{2^{d+k}(2\ell_1+1)+j}$

basis, and has it entries overwritten by the coefficients of the polynomial on the output basis. The subvectors of these vectors that are passed to Algorithms 6 and 7 can be represented in practice by offset and stride parameters. In doing so, only $O(\log \ell)$ field elements are required to be stored in auxiliary space by either algorithm, which includes the storage of the precomputed elements discussed in Section 5.2.

Theorem 5.8. *Algorithms 8 and 9 are correct.*

Proof. Correctness is only proved for Algorithm 8, since the proof for Algorithm 9 follows along similar lines. Suppose that Algorithm 8 is called on $\ell \in \{1,\dots,2^n\}$ and $(a_i)_{0\le i<\ell}$, with $a_i = f_i \in \mathbb{F}$ for $i \in \{0,\dots,\ell-1\}$. Then it is clear that the algorithm produces the correct output if $\ell = 1$, since $X_0 = 1$ and the algorithm does not modify the entries of $(a_i)_{0\le i<\ell}$ in this case. Therefore, assume that $\ell > 1$. Let $f = \sum_{i=0}^{\ell-1} f_i x^i$, and $f_0^{(t,k)}, \dots, f_{\ell-1}^{(t,k)} \in \mathbb{F}$ satisfy (5.2) for $t \in \{0,\dots,m-1\}$ and $k \in \{0,\dots,\lceil n/d_t \rceil - 1\}$, which exist and are unique by Lemma 5.1.

Lines 1 to 4 of the algorithm multiply a_i by β_0^i for $i \in \{1,\dots,\ell-1\}$ if $\beta_0 \neq 1$. If $\ell \le 2$, then the remaining lines of the algorithm have no effect on the vector $(a_i)_{0\le i<\ell}$, since $\lceil (\log_2 \ell)/d_t \rceil < 2$ for $t \in \{0,\dots,m-1\}$. As $X_0 = 1$ and $X_0 = x/\beta_0$, it follows that the algorithm produces the correct output if $\ell \le 2$. Therefore, assume that $\ell > 2$. Then $q_s < \ell \le q_{s+1}$ for some $s \in \{0,\dots,m-1\}$, and properties (1) and (4) of Lemma 5.1 imply that $a_i = f_i^{(m-1,0)} = \cdots = f_i^{(s,0)}$ for $i \in \{0,\dots,\ell-1\}$ after Lines 1 to 4 of the algorithm have been performed.

Suppose that during the algorithm, Line 7 is reached and the entries of $(a_i)_{0\le i<\ell}$ satisfy $a_i = f_i^{(t,k)}$ for $i \in \{0,\dots,\ell-1\}$, where $t \in \{0,\dots,m-1\}$ and $k \in \{0,\dots,\lceil (\log_2 \ell)/d_t \rceil - 2\}$ are the current values of the loop counter variables in Lines 5 and 6. If d_{t+1}/d_t divides $k+1$, then Lines 7

Algorithm 8 MonomialToLCH$(\ell, (a_i)_{0\le i<\ell})$

Input: $\ell \in \{1, \dots, 2^n\}$; and the vector $(a_i)_{0\le i<\ell}$ such that $a_i = f_i \in \mathbb{F}$ for $i \in \{0, \dots, \ell-1\}$.
Output: $a_i = h_i \in \mathbb{F}$ for $i \in \{0, \dots, \ell-1\}$ such that $\sum_{i=0}^{\ell-1} h_i X_i = \sum_{i=0}^{\ell-1} f_i x^i$.

1: **if** $\beta_0 \neq 1$ **and** $\ell > 1$ **then**
2: $\quad c \leftarrow \beta_0, a_1 \leftarrow ca_1$
3: $\quad$ **for** $i = 2, \dots, \ell-1$ **do**
4: $\quad\quad c \leftarrow \beta_0 c, a_i \leftarrow ca_i$
5: **for** $t = m-1, m-2, \dots, 0$ **do**
6: $\quad$ **for** $k = 0, \dots, \lceil(\log_2 \ell)/d_t\rceil - 2$ **do**
7: $\quad\quad$ **if** $d_{t+1}/d_t \nmid k+1$ **then**
8: $\quad\quad\quad$ **for** $j \in J_{t,k,\ell}$ **do**
9: $\quad\quad\quad\quad$ TaylorExpansion$(d_t, |I_{t,k,\ell-j}|, (a_{iq_t^k+j})_{0\le i<|I_{t,k,\ell-j}|})$
10: $\quad\quad\quad$ **if** $\delta_{t,k} \neq 1$ **then**
11: $\quad\quad\quad\quad u' \leftarrow \lceil |I_{t,k,\ell}|/q_t\rceil - 1, v' \leftarrow |I_{t,k,\ell}| - u'q_t, c \leftarrow \delta_{t,k}$
12: $\quad\quad\quad\quad$ **for** $u = 1, \dots, u'-1$ **do**
13: $\quad\quad\quad\quad\quad$ **for** $v = 0, \dots, q_t - 1$ **do**
14: $\quad\quad\quad\quad\quad\quad$ **for** $j \in J_{t,k,\ell-(uq_t+v)q_t^k}$ **do**
15: $\quad\quad\quad\quad\quad\quad\quad a_{(uq_t+v)q_t^k+j} \leftarrow ca_{(uq_t+v)q_t^k+j}$
16: $\quad\quad\quad\quad\quad c \leftarrow \delta_{t,k} c$
17: $\quad\quad\quad\quad$ **for** $v = 0, \dots, v'-1$ **do**
18: $\quad\quad\quad\quad\quad$ **for** $j \in J_{t,k,\ell-(u'q_t+v)q_t^k}$ **do**
19: $\quad\quad\quad\quad\quad\quad a_{(u'q_t+v)q_t^k+j} \leftarrow ca_{(u'q_t+v)q_t^k+j}$

to 19 have no effect on the vector, while property (2) of Lemma 5.1 implies that its entries satisfy $a_i = f_i^{(t,k+1)}$ for $i \in \{0, \dots, \ell-1\}$. If d_{t+1}/d_t does not divide $k+1$, then after Lines 7 to 9 have been performed, the entries of $(a_i)_{0\le i<\ell}$ satisfy

$$\sum_{i=0}^{|I_{t,k,\ell-j}|-1} f_{iq_t^k+j}^{(t,k)} x^i = \sum_{i=0}^{|I_{t,k,\ell-j}|-1} a_{iq_t^k+j} x^{i-q_t\lfloor i/q_t\rfloor} (x^{q_t} - x)^{\lfloor i/q_t\rfloor} \quad \text{for } j \in J_{t,k,\ell}.$$

If $\delta_{t,k} \neq 1$, then Lines 10 to 19 subsequently multiply $a_{iq_t^k+j}$ by $\delta_{t,k}^{\lfloor i/q_t\rfloor}$ for $i \in I_{t,k,\ell}$ and $j \in J_{t,k,\ell-iq_t^k}$ such that $i \geq q_t$, after-which property (3) of Lemma 5.1 implies that $a_i = f_i^{(t,k+1)}$ for $i \in \{0, \dots, \ell-1\}$. Therefore, regardless of whether d_{t+1}/d_t divides $k+1$, if the entries of $(a_i)_{0\le i<\ell}$ satisfy $a_i = f_i^{(t,k)}$ for $i \in \{0, \dots, \ell-1\}$ upon reaching Line 7, then $a_i = f_i^{(t,k+1)}$ for $i \in \{0, \dots, \ell-1\}$ after Lines 7 to 19 have been performed. In particular, if $t \neq 0$ and $k = \lceil(\log_2 \ell)/d_t\rceil - 2$, then property (4) of Lemma 5.1 implies that $a_i = f_i^{(t-1,0)}$ for $i \in \{0, \dots, \ell-1\}$ after Lines 7 to 19 have been performed. As $a_i = f_i^{(t,k)}$ for $i \in \{0, \dots, \ell-1\}$ the first time the algorithm reaches Line 7, which happens for $t = s$ and $k = 0$, it follows that the algorithm terminates with $a_i = f_i^{(0,\lceil(\log_2 \ell)/d_0\rceil-1)}$ for $i \in \{0, \dots, \ell-1\}$. Property (5) of Lemma 5.1 implies that this is the correct output. □

Remark 5.9. Generalising arguments of Gao and Mateer (2010, Appendix A) shows that if β is a Cantor basis and $d < n$ is a power of two, then $X_{2^{d(k+1)}} = X_{2^{dk}}^{2^d} - X_{2^{dk}}$ for $k \in \{0, \dots, \lceil n/d\rceil - 2\}$. By using this property in place of Lemma 5.4 in the proof of Lemma 5.1, it is possible to show that

Algorithm 9 LCHToMonomial$(\ell, (a_i)_{0\le i<\ell})$

Input: $\ell \in \{1, \dots, 2^n\}$; and the vector $(a_i)_{0\le i<\ell}$ such that $a_i = h_i \in \mathbb{F}$ for $i \in \{0, \dots, \ell-1\}$.
Output: $a_i = f_i \in \mathbb{F}$ for $i \in \{0, \dots, \ell-1\}$ such that $\sum_{i=0}^{\ell-1} f_i x^i = \sum_{i=0}^{\ell-1} h_i X_i$.

for $t = 0, \dots, m-1$ **do**
  for $k = \lceil (\log_2 \ell)/d_t \rceil - 2, \dots, 0$ **do**
    if $d_{t+1}/d_t \nmid k+1$ **then**
      if $\delta_{t,k} \neq 1$ **then**
        $u' \leftarrow \lceil |I_{t,k,\ell}|/q_t \rceil - 1$, $v' \leftarrow |I_{t,k,\ell}| - u'q_t$, $c \leftarrow (1/\delta_{t,k})$
        for $u = 1, \dots, u'-1$ **do**
          for $v = 0, \dots, q_t - 1$ **do**
            for $j \in J_{t,k,\ell-(uq_t+v)q_t^k}$ **do**
              $a_{(uq_t+v)q_t^k+j} \leftarrow c a_{(uq_t+v)q_t^k+j}$
          $c \leftarrow (1/\delta_{t,k})c$
        for $v = 0, \dots, v'-1$ **do**
          for $j \in J_{t,k,\ell-(u'q_t+v)q_t^k}$ **do**
            $a_{(u'q_t+v)q_t^k+j} \leftarrow c a_{(u'q_t+v)q_t^k+j}$
      for $j \in J_{t,k,\ell}$ **do**
        InverseTaylorExpansion$(d_t, |I_{t,k,\ell-j}|, (a_{iq_t^k+j})_{0\le i<|I_{t,k,\ell-j}|})$
if $\beta_0 \neq 1$ **and** $\ell > 1$ **then**
  $c \leftarrow (1/\beta_0)$, $a_1 \leftarrow c a_1$
  for $i = 2, \dots, \ell-1$ **do**
    $c \leftarrow (1/\beta_0)c$, $a_i \leftarrow c a_i$

Algorithms 8 and 9 produce the correct output when β is a Cantor basis if one takes $m = \lceil \log_2 n \rceil$ and $d_t = 2^t$ for $t \in \{0, \dots, m\}$. The substitution is necessary as the requirements on the quotients $\beta_i/\beta_{d_t\lfloor i/d_t \rfloor}$ may not be met in this case. For example, if β is a Cantor basis with $n \geq 4$, then $\beta_3 \in \mathbb{F}_{2^4} \setminus \mathbb{F}_{2^2}$ (Gao and Mateer, 2010, Lemma 3) and, thus, $\beta_3/\beta_{2\lfloor 3/2 \rfloor} = \beta_3/\beta_2 = 1/(\beta_3+1) \notin \mathbb{F}_{2^2}$. For the special case of $\ell = 2^n$, the algorithms of Lin et al. (2016b) are recovered, but expressed as nonrecursive algorithms.

5.5. Complexity

The following theorem bounds the number of operations performed by Algorithms 8 and 9.

Theorem 5.10. *If $q_s < \ell \leq q_{s+1}$ for some $s \in \{0, \dots, m-1\}$, then Algorithms 8 and 9 perform at most*

$$\frac{1}{2}\left\lfloor \frac{\ell}{2} \right\rfloor \left(\lceil \log_2 \ell \rceil \left(\left\lceil \frac{\log_2 \ell}{d_s} \right\rceil - 1 \right) + \sum_{t=0}^{s-1} d_t \left\lceil \frac{\log_2 \ell}{d_t} \right\rceil \left(\frac{d_{t+1}}{d_t} - 1 \right) \right)$$

additions in $\mathbb{F}$, and at most $(\ell-1)(\lceil \log_2 \ell \rceil + 1) - 1$ multiplications in $\mathbb{F}$.

Theorem 5.10 is proved in Section 5.6. Recall that for a generic choice of β, the algorithms of Lin et al. (2016b) allow polynomials in $\mathbb{F}[x]_{2^n}$ to be converted between the LCH and monomial bases with $O(2^n n^2)$ additions and $O(2^n n)$ multiplications. For comparison, the corresponding parameters for Algorithms 8 and 9 are $m = 1$ and $\ell = 2^n$. For these parameters, the algorithms roughly reduce to their counterparts in (Lin et al., 2016b), with both sets of algorithms performing

identical patterns of Taylor expansions and inverse Taylor expansions. These similarities result in Algorithms 8 and 9 achieving the same asymptotic complexity for the generic case as the algorithms of Lin et al., as confirmed by the following corollary of Theorem 5.10.

Corollary 5.11. *If $m = 1$, then Algorithms 8 and 9 perform at most $\lfloor \ell/2 \rfloor \binom{\lceil \log_2 \ell \rceil}{2}$ additions in $\mathbb{F}$, and at most $(\ell - 1)(\lceil \log_2 \ell \rceil + 1)$ multiplications in $\mathbb{F}$.*

Proof. Algorithms 8 and 9 perform no multiplications if $\ell = 1$, at most one multiplication if $\ell = 2$, and no additions for $\ell \leq 2$. Thus, the bounds hold for $\ell \leq 2$. If $m = 1$ and $\ell > 2$, then Theorem 5.10 can be applied with $s = 0$, which yields the stated bounds. □

Theorem 5.10 supports our earlier claim, made directly after Lemma 5.1, that each subfield of degree less than $\log_2 \ell$ in the tower (5.1) contributes to the performance of our algorithms by reducing the number of additions and multiplications they perform. The following corollary demonstrates the cumulative effect of these contributions by showing that for each $c \geq 2$, Algorithms 8 and 9 perform $O(\ell(\log \ell)(\log\log \ell))$ additions for towers such that $d_{t+1}/d_t \leq c$ for $t \in \{0, \ldots, m-1\}$, matching the bound obtained by Lin et al. (2016b) for Cantor bases.

Corollary 5.12. *Let $b, c \in \mathbb{N}$ such that $2 \leq b \leq d_{t+1}/d_t \leq c$ for $t \in \{0, \ldots, m-1\}$. Then Algorithms 8 and 9 perform at most*

$$\frac{c-1}{2}\left\lfloor \frac{\ell}{2} \right\rfloor \left(\lceil \log_2(\ell/2) \rceil \left(\lceil \log_b \log_2 \max(\ell, 2) \rceil + \frac{1}{b-1} \right) + 1 - \frac{1}{b-1} \right)$$

additions in $\mathbb{F}$.

Proof. Let $b, c \in \mathbb{N}$ such that $2 \leq b \leq d_{t+1}/d_t \leq c$ for $t \in \{0, \ldots, m-1\}$. Then the bound holds trivially if $\ell \leq 2$, since neither algorithm will perform any additions. If $\ell > 2$, then $q_s < \ell \leq q_{s+1}$ for some $s \in \{0, \ldots, m-1\}$. Thus, $s \leq \lceil \log_b \log_2 \max(\ell, 2) \rceil - 1$, $\lceil (\log_2 \ell)/d_s \rceil \leq c$ and

$$\sum_{t=0}^{s-1} d_t \left\lceil \frac{\log_2 \ell}{d_t} \right\rceil \leq s\left(\lceil \log_2 \ell \rceil - 1\right) + \sum_{t=0}^{s-1} d_t \leq s\lceil \log_2(\ell/2) \rceil + \frac{d_s - 1}{b-1} \leq s\lceil \log_2(\ell/2) \rceil + \frac{\lceil \log_2(\ell/2) \rceil - 1}{b-1}.$$

Substituting these inequalities into the bound of Theorem 5.10 then completes the proof. □

Similar to the algorithms of Lin et al. for Cantor bases, and with equally strong requirements on the field $\mathbb{F}$, the following proposition shows that multiplications are eliminated from Algorithms 8 and 9 by a special family of β.

Proposition 5.13. *If $\beta_0 = 1$, $d_{t+1}/d_t = 2$ and $\mathrm{Tr}_{\mathbb{F}_{q_{t+1}}/\mathbb{F}_{q_t}}(\beta_{d_t(k+1)}/\beta_{d_t k}) = 1$ for $t \in \{0, \ldots, m-1\}$ and even $k \in \{0, \ldots, \lceil (\log_2 \ell)/d_t \rceil - 2\}$, then Algorithms 8 and 9 perform no multiplications in $\mathbb{F}$.*

Proof. If $\beta_0 = 1$, then Lines 1 to 4 of Algorithm 8 and Lines 16 to 19 of Algorithm 9 perform no multiplications. If $d_{t+1}/d_t = 2$ and $\mathrm{Tr}_{\mathbb{F}_{q_{t+1}}/\mathbb{F}_{q_t}}(\beta_{d_t(k+1)}/\beta_{d_t k}) = 1$ for $t \in \{0, \ldots, m-1\}$ and even $k \in \{0, \ldots, \lceil (\log_2 \ell)/d_t \rceil - 2\}$, then Lines 5 to 19 of Algorithm 8 and Lines 1 to 15 of Algorithm 9 perform no multiplications, since property (3) of Lemma 5.1 implies that $\delta_{t,k} = 1$ for $t \in \{0, \ldots, m-1\}$ and $k \in \{0, \ldots, \lceil (\log_2 \ell)/d_t \rceil - 2\}$ such that d_{t+1}/d_t does not divide $k+1$. □

Remark 5.14. If $d_{t+1}/d_t = 2$ for some $t \in \{0, \ldots, m-1\}$, then property (3) of Lemma 5.1 implies that $\delta_{t,k} \in \mathbb{F}_{q_t}$ for even $k \in \{0, \ldots, \lceil n/d \rceil - 2\}$. Consequently, if some of these elements cannot

be forced to equal one, then it may still be possible to reduce the cost of the corresponding multiplications in Lines 5 to 19 of Algorithm 8 (similarly, Lines 1 to 15 of Algorithm 9) by choosing a representation of the elements of the field that lowers the cost of multiplying by elements of the subfield $\mathbb{F}_{q_t}$. Such optimisations have previously been shown to be beneficial in practice, particularly for multiplications by elements of small subfields, by Bernstein and Chou (2014) and Chen et al. (2017a).

We end the section with an example of the potential benefits offered by Algorithms 8 and 9 in a fixed field. As noted earlier in the section, the algorithms roughly reduce to those of Lin et al. (2016b) when $m = 1$. Thus, comparing Algorithms 8 and 9 against themselves for this case provides some indication of the benefits afforded by the techniques of this section. The example also demonstrates the reduction in multiplicative complexity afforded by the use of additional subfields, which is not exhibited by the bound of Theorem 5.10

Example 5.15. Under the assumption that $\mathbb{F}_{2^{12}} \subseteq \mathbb{F}$, Figure 1 displays the relative number of additions performed by Algorithms 8 and 9 for each choice of the tower (5.1) such that $d_m = 12$ and $m \geq 2$, given as a fraction of the number performed for the trivial choice of tower $\mathbb{F}_2 \subset \mathbb{F}_{2^{12}}$, as the polynomial length ℓ ranges over $\{512, \ldots, 4096\}$. As can be seen in the figure, the fraction of additions performed for each tower converges to $(\sum_{t=0}^{m-1}(d_{t+1}/d_t - 1))/(d_m - 1)$ as ℓ approaches 4096, since the bound of Theorem 5.10 is attained for $\ell = q_m$.

Figure 1 also contains two plots which display the relative number of multiplications performed by Algorithms 8 and 9 for each tower, once again given as a fraction of the number performed for the trivial choice of tower $\mathbb{F}_2 \subset \mathbb{F}_{2^{12}}$. To emphasise the influence of the choice of tower, it is assumed that $\beta_0 = 1$ in all cases, since the number of multiplications performed by Lines 1 to 4 of Algorithm 8 and Lines 16 to 19 of Algorithm 9 is independent of the tower. In the first of the two plots, it is assumed that $\delta_{t,k}$ is never equal to one. In the second plot, it is assumed that $\delta_{t,k}$ is equal to one if and only if $d_{t+1}/d_t = 2$ and k is even, in order to demonstrate the benefit of choosing β so that $\mathrm{Tr}_{\mathbb{F}_{q_{t+1}}/\mathbb{F}_{q_t}}(\beta_{d_t(k+1)}/\beta_{d_t k}) = 1$ for $t \in \{0, \ldots, m-1\}$ and $k \in \{0, \ldots, \lceil(\log_2 \ell)/d_t\rceil - 2\}$ such that $d_{t+1}/d_t = 2$ and k is even.

5.6. Proof of Theorem 5.10

We split the proof of Theorem 5.10 into five lemmas, with the first four lemmas dedicated to bounding the number of additions performed by Algorithms 8 and 9.

Lemma 5.16. *If $q_s < \ell \leq q_{s+1}$ for some $s \in \{0, \ldots, m-1\}$, then Algorithms 8 and 9 perform at most*

$$\sum_{t=0}^{s} \sum_{\substack{k=0 \\ d_{t+1}/d_t \nmid k+1}}^{\lceil(\log_2 \ell)/d_t\rceil - 2} \sum_{j \in J_{t,k,\ell}} \left\lfloor \frac{|I_{t,k,\ell-j}|}{2} \right\rfloor \left\lceil \log_2 \left\lceil \frac{|I_{t,k,\ell-j}|}{q_t} \right\rceil \right\rceil$$

additions in $\mathbb{F}$.

Proof. Additions are only performed by Line 9 of Algorithm 8 and Line 15 of Algorithm 9, with Lemma 5.7 implying that each line performs at most $\lfloor |I_{t,k,\ell-j}|/2 \rfloor \lceil \log_2 \lceil |I_{t,k,\ell-j}|/q_t \rceil \rceil$ additions. Thus, the number of additions performed by either algorithm is in-turn bounded by the sum of these bounds for $t \in \{0, \ldots, m-1\}$ and $k \in \{0, \ldots, \lceil(\log_2 \ell)/d_t\rceil - 2\}$ such that $d_{t+1}/d_t \nmid k+1$. If there exists $s \in \{0, \ldots, m-1\}$ such that $q_s < \ell \leq q_{s+1}$, then the resulting bound is equal to that of the lemma, since it follows that $\lceil(\log_2 \ell)/d_t\rceil \leq \lceil d_{s+1}/d_t \rceil < 2$ for $t \in \{s+1, \ldots, m-1\}$. □

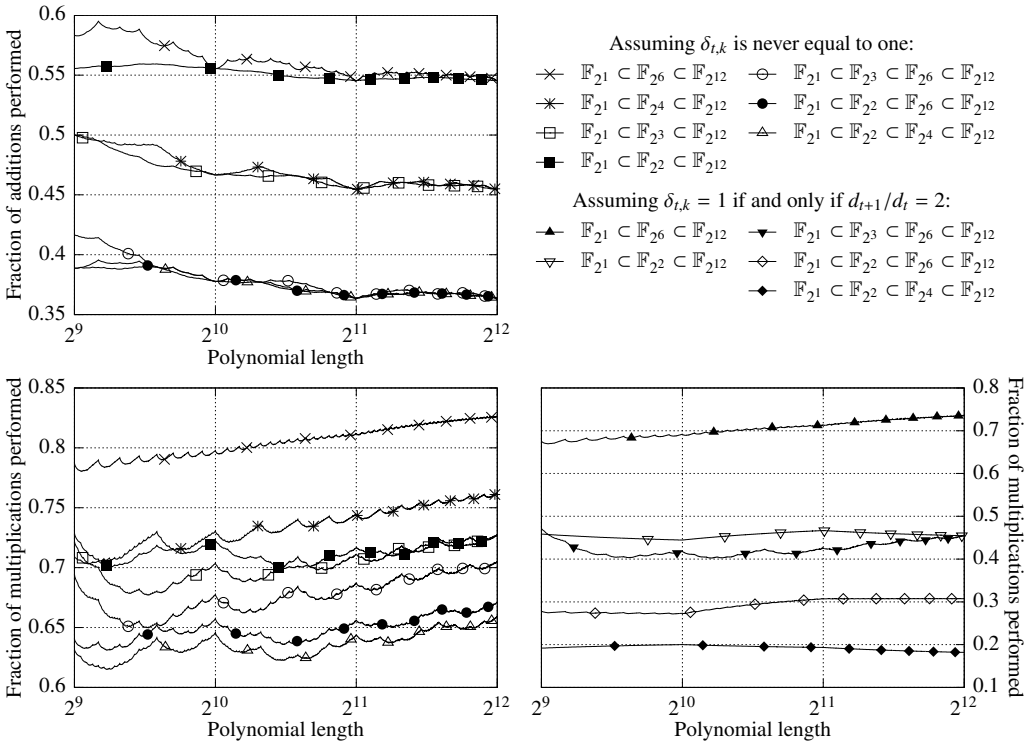

Figure 1: Relative number of operations performed by Algorithms 8 and 9 for Example 5.15.

We now bound the inner-most sums of the bound from Lemma 5.16, before working our way out to obtain the bound of Theorem 5.10.

Lemma 5.17. *If $\ell \in \{1, \dots, 2^n\}$, $t \in \{0, \dots, m-1\}$ and $k \in \{0, \dots, \lceil (\log_2 \ell)/d_t \rceil - 2\}$, then*

$$\sum_{j \in J_{t,k,\ell}} \left\lfloor \frac{|I_{t,k,\ell-j}|}{2} \right\rfloor \left\lceil \log_2 \left\lceil \frac{|I_{t,k,\ell-j}|}{q_t} \right\rceil \right\rceil \leq \frac{1}{2} q_{t+1}^{e_{t,k}} z \left(d_{t+1} e_{t,k} - d_t(k+1)\right) + \left\lfloor \frac{r}{2} \right\rfloor \left\lceil \log_2 \left\lceil \frac{r}{q_t^{k+1}} \right\rceil \right\rceil,$$

where $z = \lceil \ell / q_{t+1}^{e_{t,k}} \rceil - 1$ and $r = \ell - z q_{t+1}^{e_{t,k}}$.

Proof. Suppose that $\ell \in \{1, \dots, 2^n\}$, $t \in \{0, \dots, m-1\}$ and $k \in \{0, \dots, \lceil (\log_2 \ell)/d_t \rceil - 2\}$. Let $z = \lceil \ell / q_{t+1}^{e_{t,k}} \rceil - 1$ and $r = \ell - z q_{t+1}^{e_{t,k}}$. Then, as $r \geq 1$, $j \in J_{t,k,\ell}$ if and only if $j = j_1 q_{t+1}^{e_{t,k}} + j_0$ for some $j_1 \in \{0, \dots, z\}$ and $j_0 \in \{0, \dots, \min(q_t^k, \ell - j_1 q_{t+1}^{e_{t,k}}) - 1\}$. Moreover, if $j \in J_{t,k,\ell}$ is written in this form, then $|I_{t,k,\ell-j}| = q_{t+1}^{e_{t,k}} / q_t^k$ if $j_1 < z$, and $|I_{t,k,\ell-j}| = |I_{t,k,r-j_0}|$ if $j_1 = z$. Thus, the bound of the lemma holds if and only if

$$\sum_{j=0}^{\min(q_t^k, r)-1} \left\lfloor \frac{|I_{t,k,r-j}|}{2} \right\rfloor \left\lceil \log_2 \left\lceil \frac{|I_{t,k,r-j}|}{q_t} \right\rceil \right\rceil \leq \left\lfloor \frac{r}{2} \right\rfloor \left\lceil \log_2 \left\lceil \frac{r}{q_t^{k+1}} \right\rceil \right\rceil. \tag{5.6}$$

To prove this inequality, we may assume that $r \geq q_t^k$, since otherwise both sides of the inequality are zero. Let $u = \lfloor r / q_t^k \rfloor$ and $v = r - u q_t^k$. Then, as $r \leq q_{t+1}^{e_{t,k}}$, $|I_{t,k,r-j}| = u + \lceil (v-j)/q_t^k \rceil$ for

$j \in \{0, \dots, q_t^k - 1\}$. Consequently, the left-hand side of (5.6) is equal to

$$v\left\lfloor \frac{u+1}{2} \right\rfloor \left\lceil \log_2 \left\lceil \frac{u+1}{q_t} \right\rceil \right\rceil + \left(q_t^k - v\right) \left\lfloor \frac{u}{2} \right\rfloor \left\lceil \log_2 \left\lceil \frac{u}{q_t} \right\rceil \right\rceil.$$

We have $u = r/q_t^k$ if $v = 0$, and $u + 1 = \lceil r/q_t^k \rceil$ if $v \neq 0$. Thus, $\lceil u/q_t \rceil = \lceil r/q_t^{k+1} \rceil$ if $v = 0$, and $\lceil (u+1)/q_t \rceil = \lceil r/q_t^{k+1} \rceil$ if $v \neq 0$. It follows that the left-hand side of (5.6) is less than or equal to

$$\left\lfloor \frac{v(u+1) + (q_t^k - v)u}{2} \right\rfloor \left\lceil \log_2 \left\lceil \frac{r}{q_t^{k+1}} \right\rceil \right\rceil = \left\lfloor \frac{r}{2} \right\rfloor \left\lceil \log_2 \left\lceil \frac{r}{q_t^{k+1}} \right\rceil \right\rceil.$$

Therefore, (5.6) holds, which completes the proof of the lemma. □

Combining the following two lemmas with Lemma 5.16 completes the proof of the addition bound of Theorem 5.10.

Lemma 5.18. *If $\ell \in \{1, \dots, 2^n\}$ and $t \in \{0, \dots, m-1\}$ such that $q_{t+1} < \ell$, then*

$$\sum_{\substack{k=0 \\ d_{t+1}/d_t \nmid k+1}}^{\lceil (\log_2 \ell)/d_t \rceil - 2} \sum_{j \in J_{t,k,\ell}} \left\lfloor \frac{|I_{t,k,\ell-j}|}{2} \right\rfloor \left\lceil \log_2 \left\lceil \frac{|I_{t,k,\ell-j}|}{q_t} \right\rceil \right\rceil \leq \left\lfloor \frac{\ell}{2} \right\rfloor \frac{d_t}{2} \left\lceil \frac{\log_2 \ell}{d_t} \right\rceil \left(\frac{d_{t+1}}{d_t} - 1 \right).$$

Proof. Suppose that $\ell \in \{1, \dots, 2^n\}$ and $t \in \{0, \dots, m-1\}$ such that $q_{t+1} < \ell$. Then, for $k \in \{0, \dots, \lceil (\log_2 \ell)/d_t \rceil - 2\}$, the integers $z = \lceil \ell / q_{t+1}^{e_{t,k}} \rceil - 1$ and $r = \ell - z q_{t+1}^{e_{t,k}}$ satisfy $r \leq q_{t+1}^{e_{t,k}}$ and $z q_{t+1}^{e_{t,k}}/2 + \lfloor r/2 \rfloor = \lfloor \ell/2 \rfloor$. Thus, Lemma 5.17 implies that

$$\sum_{j \in J_{t,k,\ell}} \left\lfloor \frac{|I_{t,k,\ell-j}|}{2} \right\rfloor \left\lceil \log_2 \left\lceil \frac{|I_{t,k,\ell-j}|}{q_t} \right\rceil \right\rceil \leq \left\lfloor \frac{\ell}{2} \right\rfloor \left(d_{t+1} e_{t,k} - d_t(k+1) \right) \tag{5.7}$$

for $k \in \{0, \dots, \lceil (\log_2 \ell)/d_t \rceil - 2\}$. If $k \in \{0, \dots, \lceil (\log_2 \ell)/d_t \rceil - 2\}$ such that d_{t+1}/d_t does not divide $k+1$, then $k = k_1(d_{t+1}/d_t) + k_0$ for nonnegative integers $k_1 < (d_t/d_{t+1}) \lceil (\log_2 \ell)/d_t \rceil$ and $k_0 < d_{t+1}/d_t - 1$. Moreover, when k is written in this form, we have

$$d_{t+1} e_{t,k} - d_t(k+1) = d_{t+1} \left\lceil \frac{d_t(k_0+1)}{d_{t+1}} \right\rceil - d_t(k_0+1) = d_{t+1} - d_t(k_0+1).$$

As $(d_t/d_{t+1}) \lceil (\log_2 \ell)/d_t \rceil > (d_t/d_{t+1}) \lceil d_{t+1}/d_t \rceil = 1$, it follows by substituting into (5.7) and summing the resulting inequalities that

$$\begin{aligned} \sum_{\substack{k=0 \\ d_{t+1}/d_t \nmid k+1}}^{\lceil (\log_2 \ell)/d_t \rceil - 2} \sum_{j \in J_{t,k,\ell}} \left\lfloor \frac{|I_{t,k,\ell-j}|}{2} \right\rfloor \left\lceil \log_2 \left\lceil \frac{|I_{t,k,\ell-j}|}{q_t} \right\rceil \right\rceil &\leq \left\lfloor \frac{\ell}{2} \right\rfloor \frac{d_t}{d_{t+1}} \left\lceil \frac{\log_2 \ell}{d_t} \right\rceil \sum_{k_0=0}^{d_{t+1}/d_t - 2} d_{t+1} - d_t(k_0+1) \\ &= \left\lfloor \frac{\ell}{2} \right\rfloor \frac{d_t}{2} \left\lceil \frac{\log_2 \ell}{d_t} \right\rceil \left(\frac{d_{t+1}}{d_t} - 1 \right), \end{aligned}$$

which completes the proof of the lemma. □

Lemma 5.19. *If $\ell \in \{1, \dots, 2^n\}$ and $s \in \{0, \dots, m-1\}$ such that $q_s < \ell \leq q_{s+1}$, then*

$$\sum_{\substack{k=0 \\ d_{s+1}/d_s \nmid k+1}}^{\lceil (\log_2 \ell)/d_s \rceil - 2} \sum_{j \in J_{s,k,\ell}} \left\lfloor \frac{|I_{s,k,\ell-j}|}{2} \right\rfloor \left\lceil \log_2 \left\lceil \frac{|I_{s,k,\ell-j}|}{q_s} \right\rceil \right\rceil \leq \left\lfloor \frac{\ell}{2} \right\rfloor \frac{\lceil \log_2 \ell \rceil}{2} \left(\left\lceil \frac{\log_2 \ell}{d_s} \right\rceil - 1 \right).$$

Proof. Suppose that $\ell \in \{1, \dots, 2^n\}$ and $s \in \{0, \dots, m-1\}$ satisfy $q_s < \ell \leq q_{s+1}$. Then, for $k \in \{0, \dots, \lceil (\log_2 \ell)/d_s \rceil - 2\}$, we have $z = \lceil \ell / q_{s+1}^{e_{s,k}} \rceil - 1 = 0$ and $r = \ell - z q_{s+1}^{e_{s,k}} = \ell > q_s^{k+1}$. Thus, Lemma 5.17 implies that

$$\sum_{j \in J_{s,k,\ell}} \left\lfloor \frac{|I_{s,k,\ell-j}|}{2} \right\rfloor \left\lceil \log_2 \left\lceil \frac{|I_{s,k,\ell-j}|}{q_s} \right\rceil \right\rceil \leq \left\lfloor \frac{\ell}{2} \right\rfloor \left\lceil \log_2 \left\lceil \frac{\ell}{q_s^{k+1}} \right\rceil \right\rceil = \left\lfloor \frac{\ell}{2} \right\rfloor (\lceil \log_2 \ell \rceil - d_t(k+1))$$

for $k \in \{0, \dots, \lceil (\log_2 \ell)/d_s \rceil - 2\}$. Moreover, $d_{s+1}/d_s \nmid k+1$ for $k \in \{0, \dots, \lceil (\log_2 \ell)/d_s \rceil - 2\}$, since $\lceil (\log_2 \ell)/d_s \rceil - 1 < d_{s+1}/d_s$. It follows that

$$\begin{aligned} \sum_{\substack{k=0 \\ d_{s+1}/d_s \nmid k+1}}^{\lceil (\log_2 \ell)/d_s \rceil - 2} \sum_{j \in J_{s,k,\ell}} \left\lfloor \frac{|I_{s,k,\ell-j}|}{2} \right\rfloor \left\lceil \log_2 \left\lceil \frac{|I_{s,k,\ell-j}|}{q_s} \right\rceil \right\rceil &\leq \left\lfloor \frac{\ell}{2} \right\rfloor \left(\left\lceil \frac{\log_2 \ell}{d_s} \right\rceil - 1 \right) \left(\lceil \log_2 \ell \rceil - \frac{d_s}{2} \left\lceil \frac{\log_2 \ell}{d_s} \right\rceil \right) \\ &\leq \left\lfloor \frac{\ell}{2} \right\rfloor \left(\left\lceil \frac{\log_2 \ell}{d_s} \right\rceil - 1 \right) \frac{\lceil \log_2 \ell \rceil}{2}, \end{aligned}$$

which completes the proof of the lemma. □

We now complete the proof of Theorem 5.10 by bounding the number of multiplications performed by Algorithms 8 and 9.

Lemma 5.20. *If $\ell > 1$, then Algorithms 8 and 9 perform at most $(\ell - 1)(\lceil \log_2 \ell \rceil + 1) - 1$ multiplications in $\mathbb{F}$.*

Proof. Suppose that $\ell > 1$. Then Lines 1 to 4 of Algorithm 8 perform at most $2(\ell - 1) - 1$ multiplications, while Lines 5 to 19 perform at most

$$\sum_{t=0}^{m-1} \sum_{\substack{k=0 \\ d_{t+1}/d_t \nmid k+1}}^{\lceil (\log_2 \ell)/d_t \rceil - 2} \left(\left\lceil \frac{|I_{t,k,\ell}|}{q_t} \right\rceil - 2 + \sum_{i=q_t}^{|I_{t,k,\ell}|-1} \left| J_{t,k,\ell-iq_t^k} \right| \right)$$

multiplications. The same bounds hold respectively for Lines 16 to 19 and Lines 1 to 15 of Algorithm 9. If $t \in \{0, \dots, m-1\}$ and $k \in \{0, \dots, \lceil (\log_2 \ell)/d_t \rceil - 2\}$ such that d_{t+1}/d_t does not divide $k+1$, then $\ell > q_t^{k+1}$ and $d_{t+1} e_{t,k} - d_t k > d_t$. For such t and k, it follows that $|I_{t,k,\ell}| > q_t$ and $|J_{t,k,\ell-iq_t^k}| \geq q_t^k$ for $i \in \{0, \dots, q_t - 1\}$, since $\ell - iq_t^k \geq q_t^k$ for $i \leq q_t - 1$. Thus,

$$\sum_{i=q_t}^{|I_{t,k,\ell}|-1} \left| J_{t,k,\ell-iq_t^k} \right| = \sum_{i \in I_{t,k,\ell}} \left| J_{t,k,\ell-iq_t^k} \right| - \sum_{i=0}^{q_t-1} \left| J_{t,k,\ell-iq_t^k} \right| \leq \ell - q_t^{k+1}$$

and $\lceil |I_{t,k,\ell}|/q_t \rceil - 1 \leq \lceil \lceil \ell / q_t^k \rceil / q_t \rceil - 1 \leq \ell / q_t^{k+1}$ for $t \in \{0, \dots, m-1\}$ and $k \in \{0, \dots, \lceil (\log_2 \ell)/d_t \rceil - 2\}$ such that d_{t+1}/d_t does not divide $k+1$. By combining these inequalities with (5.3), it follows that Algorithms 8 and 9 perform at most

$$(\ell - 1)(\lceil \log_2 \ell \rceil + 1) - 1 + \sum_{t=0}^{m-1} \sum_{\substack{k=0 \\ d_{t+1}/d_t \nmid k+1}}^{\lceil (\log_2 \ell)/d_t \rceil - 2} \left(\frac{\ell}{q_t^{k+1}} - q_t^{k+1} \right)$$

multiplications. Here, the double summation is equal to $(\ell / 2^{\lceil \log_2 \ell \rceil} - 1) \sum_{k=1}^{\lceil \log_2 \ell \rceil - 1} 2^k \leq 0$, since the sets $\{d_t(k+1) \mid k \in \{0, \dots, \lceil (\log_2 \ell)/d_t \rceil - 2\}, d_{t+1}/d_t \nmid k+1\}$ for $t \in \{0, \dots, m-1\}$ are pairwise disjoint and their union is $\{1, \dots, \lceil \log_2 \ell \rceil - 1\}$. □

References

Ben-Sasson, E., Bentov, I., Chiesa, A., Gabizon, A., Genkin, D., Hamilis, M., Pergament, E., Riabzev, M., Silberstein, M., Tromer, E., Virza, M., 2017. Computational integrity with a public random string from quasi-linear PCPs, in: Advances in cryptology—EUROCRYPT 2017. Part III. Springer, Cham. volume 10212 of *Lecture Notes in Comput. Sci.*, pp. 551–579.

Ben-Sasson, E., Bentov, I., Horesh, Y., Riabzev, M., 2018. Scalable, transparent, and post-quantum secure computational integrity. Cryptology ePrint Archive, Report 2018/046. `https://eprint.iacr.org/2018/046`.

Bernstein, D.J., Chou, T., 2014. Faster binary-field multiplication and faster binary-field MACs, in: Selected areas in cryptography—SAC 2014. Springer, Cham. volume 8781 of *Lecture Notes in Comput. Sci.*, pp. 92–111.

Bernstein, D.J., Chou, T., Schwabe, P., 2013. McBits: Fast constant-time code-based cryptography, in: Cryptographic Hardware and Embedded Systems—CHES 2013, Springer, Berlin. pp. 250–272.

Bitner, J.R., Ehrlich, G., Reingold, E.M., 1976. Efficient generation of the binary reflected Gray code and its applications. Comm. ACM 19, 517–521.

Bostan, A, Schost, É., 2005. Polynomial evaluation and interpolation on special sets of points. J. Complexity 21, 420–446.

Brent, R.P., Gaudry, P., Thomé, E., Zimmermann, P., 2008. Faster multiplication in GF(2)[x], in: Algorithmic number theory—ANTS 2008. Springer, Berlin. volume 5011 of *Lecture Notes in Comput. Sci.*, pp. 153–166.

Cantor, D.G., 1989. On arithmetical algorithms over finite fields. J. Combin. Theory Ser. A 50, 285–300.

Chen, M., Cheng, C., Kuo, P., Li, W., Yang, B., 2017a. Faster multiplication for long binary polynomials. `arXiv:1708.09746 [cs.SC]`.

Chen, M., Cheng, C., Kuo, P., Li, W., Yang, B., 2018. Multiplying boolean polynomials with Frobenius partitions in additive fast Fourier transform. `arXiv:1803.11301 [cs.SC]`.

Chen, M.S., Li, W.D., Peng, B.Y., Yang, B.Y., Cheng, C.M., 2018. Implementing 128-bit secure MPKC signatures. IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences E101.A, 553–569.

Chou, T., 2017. McBits revisited, in: Cryptographic Hardware and Embedded Systems—CHES 2017, Springer, Cham. pp. 213–231.

Coxon, N., 2019. Fast systematic encoding of multiplicity codes. J. Symbolic Comput. 94, 234–254.

Coxon, N., 2020. Fast Hermite interpolation and evaluation over finite fields of characteristic two. J. Symbolic Comput. 98, 270–283.

Furia, C.A., 2014. Rotation of sequences: Algorithms and proofs. `arXiv:1406.5453 [cs.LO]`.

Gao, S., Mateer, T., 2010. Additive fast Fourier transforms over finite fields. IEEE Trans. Inform. Theory 56, 6265–6272.

von zur Gathen, J., Gerhard, J., 1996. Arithmetic and factorization of polynomial over $\mathbb{F}_2$ (extended abstract), in: ISSAC '96—Proceedings of the 1996 International Symposium on Symbolic and Algebraic Computation, ACM, New York. pp. 1–9.

Gerhard, J., 2000. Modular algorithms for polynomial basis conversion and greatest factorial factorization, in: Proceedings of the Seventh Rhine Workshop on Computer Algebra, pp. 125–141.

Gries, D., Mills, H., 1981. Swapping sections. Technical Report TR 81-452. Cornell University. `https://hdl.handle.net/1813/6292`.

Harvey, D., 2009. A cache-friendly truncated FFT. Theoret. Comput. Sci. 410, 2649–2658.

Harvey, D., Roche, D.S., 2010. An in-place truncated Fourier transform and applications to polynomial multiplication, in: ISSAC 2010—Proceedings of the 2010 International Symposium on Symbolic and Algebraic Computation. ACM, New York, pp. 325–329.

van der Hoeven, J., 2004. The truncated Fourier transform and applications, in: ISSAC 2004—Proceedings of the 2004 international symposium on Symbolic and algebraic computation. ACM, New York, pp. 290–296.

van der Hoeven, J., 2005. Notes on the Truncated Fourier Transform. Technical Report 2005-5. Université Paris-Sud, Orsay, France.

van der Hoeven, J., Schost, É., 2013. Multi-point evaluation in higher dimensions. Appl. Algebra Engrg. Comm. Comput. 24, 37–52.

Knuth, D.E., 2005. The art of computer programming. Vol. 4, Fasc. 2. Addison-Wesley, Upper Saddle River, NJ.

Larrieu, R., 2017. The truncated Fourier transform for mixed radices, in: ISSAC'17—Proceedings of the 2017 ACM International Symposium on Symbolic and Algebraic Computation. ACM, New York, pp. 261–268.

Li, W.D., Chen, M.S., Kuo, P.C., Cheng, C.M., Yang, B.Y., 2018. Frobenius additive fast fourier transform, in: Proceedings of the 2018 ACM International Symposium on Symbolic and Algebraic Computation, ACM, New York, NY, USA. pp. 263–270.

Lin, S.J., Al-Naffouri, T.Y., Han, Y.S., 2016a. FFT algorithm for binary extension finite fields and its application to Reed-Solomon codes. IEEE Trans. Inform. Theory 62, 5343–5358.

Lin, S.J., Al-Naffouri, T.Y., Han, Y.S., Chung, W.H., 2016b. Novel polynomial basis with fast Fourier transform and its application to Reed–Solomon erasure codes. IEEE Trans. Inform. Theory 62, 6284–6299.

Lin, S.J., Chung, W.H., Han, Y.S., 2014. Novel polynomial basis and its application to Reed-Solomon erasure codes, in: 55th Annual IEEE Symposium on Foundations of Computer Science—FOCS 2014. IEEE Computer Soc., Los Alamitos, CA, pp. 316–325.

Markel, J., 1971. FFT pruning. IEEE Transactions on Audio and Electroacoustics 19, 305–311.

Mateer, T., 2008. Fast Fourier Transform algorithms with applications. ProQuest LLC, Ann Arbor, MI. Ph.D. thesis–Clemson University.

Shene, C.K., 1997. An analysis of two in-place array rotation algorithms. Comput. J. 40, 541–546.

Smarzewski, R., Kapusta, J., 2007. Fast Lagrange-Newton transformations. J. Complexity 23, 336–345.

Sorensen, H.V., Burrus, C.S., 1993. Efficient computation of the DFT with only a subset of input or output points. IEEE Transactions on Signal Processing 41, 1184–1200.

Wang, Y., Zhu, X., 1988. A fast algorithm for the Fourier transform over finite fields and its VLSI implementation. IEEE Journal on Selected Areas in Communications 6, 572–577.